

全国计算机技术与软件专业技术资格（水平）考试辅导用书

# 程序员考试辅导

全国计算机技术与软件专业技术资格（水平）考试办公室组编  
谢树煜 主编

清华大学出版社



全国计算机技术与软件专业技术资格（水平）考试辅导用书

TopSage.com

# 程序员考试辅导

全国计算机技术与软件专业技术资格（水平）考试办公室组编  
谢树煜 主编

清华大学出版社  
北京

[www.TopSage.com](http://www.TopSage.com)



## 全国计算机技术与软件专业资格(水平)考试真题及答案

[2008年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2008年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2007年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2007年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2006年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2006年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2009年计算机技术与软件水平考试各科目考试大纲汇总](#)

[全国计算机技术与软件专业资格\(水平\)考试真题及答案汇总](#)

[\[软考视频\]计算机技术与软件专业资格考试推荐视频教程下载汇总](#)

教材及同步辅导见下页。

# 计算机技术与软件专业技术(水平)考试指定教材及同步辅导

## 软考初级:

[程序员教程\(第二版\)2007 版 软考指定用书 高清PDF版](#)

[程序员考试辅导: 全国计算机技术与软件专业技术资格\(水平\)考试辅导用书](#)

[网络管理员教程\(第 2 版\)2007 版 软考指定用书 高清PDF版](#)

[网络管理员考试同步辅导\(计算机与网络基础知识篇\) 软考指定辅导用书](#)

[网络管理员考试同步辅导\(网络系统管理与维护篇\) 软考指定使用辅导用书](#)

## 软考中级:

[网络工程师教程\(第 2 版\) 2007 版 软考指定用书 高清PDF版](#)

[网络工程师教程 软考指定用书 高清PDF版](#)

[网络工程师考试同步辅导: 计算机与网络知识篇 软考指定用书](#)

[网络工程师考试同步辅导\(网络系统设计与管理篇\) 软考指定辅导用书](#)

[软件设计师教程\(第 2 版\) 2007 版 软考指定用书 高清PDF版](#)

[软件设计师考试同步辅导\(下午科目\) 高清PDF版](#)

[软件设计师考试同步辅导\(上午科目\) 高清PDF版](#)

[软件设计师考试考点分析与真题详解\(软件设计技术篇\)](#)



[软件设计师考试辅导：考点精讲、例题分析、强化训练 冶金工业出版](#)

[数据库系统工程师教程 软考指定用书 高清PDF版](#)

[软件评测师教程 软考指定教材 高清PDF版](#)

[信息系统管理工程师教程 软考指定用书 高清PDF版](#)

[信息系统监理师教程 软考指定用书 高清PDF版](#)

**软考高级：**

[系统分析师教程 软考指定教材 高清PDF版](#)

[系统分析师考试辅导\(2007 版\) 软考指定辅导用书 高清PDF版](#)

[系统分析师教程 PDF文字版](#)

[系统分析师经典教材 Word版](#)

[信息系统项目管理师教程 软考指定教材 高清PDF版](#)

[信息系统项目管理师辅导教程\(上下册\)](#)

[计算机专业英语教程 PDF文字版](#)

**更多计算机资料请访问：**[大家论坛计算机专区](#)

## 内 容 简 介

本书是根据中国计算机技术及软件专业技术资格(水平)考试 2004 年《程序员考试大纲》的要求,参照《程序员教程》的结构及历年软件专业资格考试试题编写的,内容紧扣考试大纲。全书共分 12 章,每章都由内容提要、例题分析、思考练习题组成。内容提要是对有关章节知识的提炼,给出考试要点和学习难点。例题分析是全书重点,着重解析考试大纲要求的基本知识及其综合应用方法。思考练习题供读者检验自己对有关内容掌握的程度。为了帮助学员提高理解程序、编制程序及软件设计的能力,本书专门增加了 C/C++ 语言程序设计一章,并在有关章节中加大了软件工程、数据结构和常用算法设计方法的比重。

本书供参加“程序员资格考试”的学员应试复习时使用,也可供大专院校及相应层次的计算机技术人员学习参考。

版权所有,翻印必究。举报电话:010-62782989 13501256678 13801310933

本书扉页为防伪纸、封面贴有清华大学出版社防伪标签,无标签者不得销售。

本书防伪标签采用特殊防伪技术,用户可通过在图案表面涂抹清水,图案消失,水干后图案复现;或将表面膜揭下,放在白纸上用彩笔涂抹,图案在白纸上再现的方法识别真伪。

### 图书在版编目(CIP)数据

程序员考试辅导 / 谢树煜主编. —北京:清华大学出版社, 2005.5

(全国计算机技术与软件专业技术资格(水平)考试辅导用书)

ISBN 7-302-10735-1

I. 程… II. 谢… III. 程序设计-工程技术人员-资格考核-自学参考资料 IV. TP311.1

中国版本图书馆 CIP 数据核字(2005)第 025999 号

出 版 者: 清华大学出版社 地 址: 北京清华大学学研大厦

<http://www.tup.com.cn> 邮 编: 100084

社 总 机: 010-62770175 客户服务: 010-62776969

组稿编辑: 柴文强

文稿编辑: 林晴佳

印 刷 者: 清华大学印刷厂

装 订 者: 北京鑫海金澳胶印有限公司

发 行 者: 新华书店总店北京发行所

开 本: 185 × 230 印张: 36.5 防伪页: 1 字数: 818 千字

版 次: 2005 年 5 月第 1 版 2005 年 9 月第 2 次印刷

书 号: ISBN 7-302-10735-1/TP · 7147

印 数: 5001 ~ 10000

定 价: 46.00 元



## 前 言

计算机技术及软件专业技术资格（水平）考试是国家人事部与信息产业部主办的国家级考试，十余年来为国家选拔和培养了十多万名合格的软件技术人才，在国内外产生很大影响，得到社会各界广泛认同。

2000 年 1 月，为了推动中日两国间信息技术的交流与合作，中国软件技术资格（水平）考试与日本信息处理技术人员考试就 IT 考试标准达成相互认证，成为我国与日本政府级的双方互相承认的软件专业技术资格考试，使软件专业技术资格考试逐步走上与国际接轨之路。

2003 年 10 月，国家人事部与信息产业部发布的 39 号文件规定，把计算机技术与软件专业技术资格考试纳入全国专业技术人员职业资格证书制度的统一规划中。通过考试取得技术资格证书的人员，表明已具备相应专业岗位工作水平和能力，用人单位可择优聘任其担任相应专业技术职务。同时决定今后不再进行相应专业技术职务任职资格的评审工作，因此这种考试既是职业资格考试，又是技术资格考试。我们相信这种以考代评的重大改革，对软件专业技术人才培养将起到巨大推动作用。

2004 年 5 月，软件专业技术资格（水平）考试办公室公布了新的考试大纲，对考试内容做了若干调整，扩大了软件工程的考试范围，增加了软件标准化与知识产权的考试要求。为了帮助广大学员深入理解考试大纲的要求，掌握有关课程的基本概念，基本内容和基本方法，进一步提高应试能力，在信息产业部计算机技术与软件专业技术资格考试办公室领导下，在清华大学出版社支持下，我们编写了《程序员考试辅导》。该书是根据两部考试办公室制定的考试大纲的要求，配合学员考试自学复习的需要编写的。书中介绍了有关专业要求的基本知识和技能，内容涵盖软件专业的核心课程。全书包括计算机系统基础知识、操作系统、数据库、多媒体、计算机网络、程序设计语言的基础知识，强化了软件工程、数据结构、常用算法设计方法的内容，增加了软件标准化和知识产权的基础知识。为了帮助学员学习 C 和 C++ 程序设计语言，本书特别增加了一章 C/C++ 语言程序设计。

考试大纲要求学员掌握的知识面很宽，考虑到学员复习的时间有限，书中针对考试大纲及教材的内容要点和学习难点作了剖析，并把重点放在例题分析上，这些题目都是作者从自己切身教学经验和历届考题中精选出来的，例题分析中对有关解题思路、解题方法，应用的基本知识和基本原理，做了详尽介绍，一定会对参加考试的学员有所启发和帮助。每章还附有思考练习题及答案，供学员自我检查练习时使用。

本书由清华大学谢树煜教授主编，全书共分 12 章，第 1 章计算机系统基础知识由

谢树煜编写,第2章操作系统基础知识由北京大学方裕教授编写,第3章数据库基础知识由北京科技大学王道平教授编写,第4章多媒体基础知识由清华大学林福宗教授编写,第5章网络基础知识由北京农业大学孙瑞志副教授编写(清华大学计算机系博士、他在网络方面有丰富的经验),第6章程序设计语言基础由北京大学丁文魁教授编写,第7章软件系统开发与运行、第8章数据结构由清华大学殷人昆教授编写,第11章知识产权基础知识由国家软件保护中心李维高级工程师编写,第9章常用算法设计方法、第10章标准化基础知识与第12章C/C++语言程序设计由清华大学孙甲松副教授完成。他们都是相关学科的专家和教授,并且有丰富的教学经验,很多老师多年来担任过软件专业技术资格和水平考试的考前培训工作,积累了很多培训经验。本书编写过程中得到清华大学出版社柴文强编辑的大力支持和帮助,特此表示感谢。

由于水平和时间有限,书中不妥之处敬请指正。

编 者

2004年12月于清华园



## 应试寄语

参加软件专业技术资格（水平）考试的学员首先必须认真阅读考试大纲。要求“熟练掌握”的内容需要特别重视，在考试试题中占的比重较大。要求“掌握”的内容是考试的重点，要求“熟悉”和“了解”的内容重要性次之，一般对待即可，但不等于不考。本书中对各部分内容要点和难点做了分析，可供读者复习时参考。

考试分上午考试和下午考试，时间都是 150 分钟，上午试题原来分为 15 个大题，每题 5 分，均为填空题，满分 75 分。2001 年以后把 75 分分拆为 75 个填空小题，每个填空考查一个知识点，这样涉及的知识面更宽一些。

上午试题都是选择填空题，按考试要求可分为两类，第一类是考查基本知识题，比较简单，概念清楚即可得分。例如 2002 年试题第（26）、（27）小题，叙述如下：

- ① 软件从一个计算机系统转换到另一个计算机系统运行的难易程度是指软件的（26）。在规定的条件下和规定的时间间隔内，软件实现其规定功能的概率称为（27）。

可选答案：（26） A. 兼容性                      B. 可移植性  
                    C. 可转换性                      D. 可接近性  
                    （27） A. 可使用性                      B. 可接近性  
    C. 可靠性                      D. 稳定性

解题分析：软件质量规定为软件产品满足规定需求的能力的各种特性。因此，定义一个软件的质量就是为该软件规定一系列质量特性。例如，把一个软件从一个计算机的系统上转换到另一个计算机系统上运行的难易程度叫可移植性；在规定条件下和规定时间内软件实现其规定的功能的概率称为可靠性。

因此，（26）小题答案选择 B，（27）小题选择答案 C。

第二类选择填空题是计算型选择题，是需要经过计算才能确定答案的，不但要求掌握有关计算方法还需计算正确，例如 2002 年试题（58）、（59）题，试题叙述如下：

- ② 某硬盘共有 9 个盘片，16 个记录面，每个记录面上有 2100 个磁道，每个磁道分为 64 个扇区，每个扇区存储 512 字节数据，则磁盘的存储容量为（58）。磁盘的位密度随着磁道从内向外（59）。

可选答案：（58） A. 590.6MB                      B. 9225MB  
                    C. 1050MB                      D. 1101MB  
                    （59） A. 减少                      B. 不变  
    C. 增加                      D. 视磁盘而定

解题分析：(58) 磁盘存储容量为扇区容量  $\times$  扇区数  $\times$  磁道数  $\times$  记录面数

$$512\text{B} \times 64 \times 2100 \times 16 = 32\text{KB} \times 2100 \times 16 = 1050\text{MB}$$

因为  $1\text{MB} = 2^{20} = 1024 \times 1024 = 1048576\text{B}$

答案应选 C。

(59) 磁盘上的每个磁道存储的二进制信息字节数都是相同的，外圈磁道周长较长，其位密度较低。

(59) 题答案选择 A。

另外还有一种考题是从若干条叙述中选择哪些叙述是正确的，这种题也可列为基本知识型选择题。

从历年考试内容看，各门课程考试比重不同，反映考试要求不同，上午试题满分 75 分，英语占 10 分，其他课程每年考题分量略有差异。

2003 年上午考试为 75 个小题，每小题 1 分，试题分布更加均匀一些，英语 10 分，计算机组织与结构 17 分，软件工程 12 分，数据结构 6 分，程序语言 6 分，数据库 6 分，操作系统 5 分，多媒体 3 分，网络 8 分，标准化和知识产权 4 分。

2004 年上半年试题分布为：英语 10 分，计算机组织与结构 11 分，软件工程 7 分，数据结构 7 分，程序语言 6 分，数据库 5 分，操作系统 11 分，多媒体 6 分，网络 9 分，标准化和知识产权 3 分。

从考试内容的重点看需注意以下几点。

软件知识方面，数据结构考查树与二叉树查找、常用排序算法，其中二叉树遍历方法及堆排序应引起注意，程序语言考查编译和解译基本知识包括语法语义分析等，操作系统涉及进程概念、PV 操作及作业管理等，软件工程涉及数据流图，软件设计方法、软件质量和软件风格、软件开发、测试等，数据库考查 SQL 关系模式、关系运算和关系代数表达式，以及数据库模型、结构、DBMS 等，多媒体主要考查数字图像，数据压缩和 RGB 原理等。

硬件知识方面，主要考查数制、码制、定点数、浮点数数据表示方法、校验码、算术逻辑运算、指令系统、寻址方式等，以及 I/O 传送控制方式、存储器和磁盘容量计算等。网络方面考查网络标准和协议知识等。

每年考题都有调整，近几年加大了软件工程，数据结构算法分析考试比重，但内容划分变化不太大，基本是围绕考试大纲命题的。

下午为程序设计考试，时间 150 分钟。

主要考查算法、数据结构以及编程能力，考查学生理解数据流图、程序流程图、C 语言编程和算法应用的能力，考查学生对程序和算法实现的理解能力。考生必须完成 5 个试题，每题 15 分，满分 75 分。

编程必须掌握 C 语言，程序员下午试题要求考生熟练掌握基本算法和数据结构，能按软件设计说明书，使用 C 语言熟练编制程序。要求掌握查找、更新、排序、合并、分



支等基本算法，熟悉迭代、插值、矩阵计算、搜索求解算法，编制分支、循环、子程序递归等程序以及输入输出文件处理方法等。

编程能力考查对程序设计语言和算法实现的理解能力。根据算法定义，算法说明注释、理解给定算法的实现方法并理解程序，一般考试给定 5 个题，有一个题考查对 C 语言基本知识的理解，其他题考查算法求解等，2002 年、2003 年考试中共有 5 个题都是必答题，若有选做题，则多为从 C、C++、Java 或 VB 中选答一题，按照要求，不要多答。

2002 年下午试题，试题 1 考查二维数组，试题 2 为字符串比较，试题 3 为插入法排序，试题 4 用古典 Eratos thenes 的算法求指定范围的素数，试题 5 采用递归法，将结点 S 插入到一个二叉排序树中。在答题技巧上，建议考生先浏览一次卷面，对整个考试做到心中有数。其次认真看清题意后再动手答题，先易后难，有把握的题先抓着，再仔细分辨没有把握的题，经过翻来覆去的比较，决定取舍，然后再对非常困难的试题，作出常识性的判断，其答对的几率还有 50%，不要轻易放弃，最后还要检查一遍有无漏答的题。

总之答题时头脑要冷静，面对难题不要惊慌失措，面对容易题不要得意忘形，忘乎所以，经过深入思考，总是可以想清楚的。

复习时，看看历届的考题，试试身手，也是有益的，但一定要有信心，要相信自己。人贵有志，只要有决心有志气，一次过不去也没什么了不起，最后的成功一定属于你。



# 目 录

<b>第 1 章 计算机系统基础知识</b>	<b>1</b>
1.1 内容提要	1
1.1.1 计算机基本组成和特性	1
1.1.2 数据表示	2
1.1.3 算术运算和逻辑运算	7
1.1.4 计算机组成原理	9
1.1.5 指令系统	13
1.1.6 计算机系统性能评价	15
1.2 例题分析	16
1.2.1 计算机的基本组成和特性	16
1.2.2 数据表示方法	20
1.2.3 运算方法	36
1.2.4 计算机组成原理	43
1.2.5 指令系统	69
1.2.6 计算机系统性能评价	74
1.3 思考练习题及答案	77
思考练习题	77
思考练习题答案	83
<b>第 2 章 操作系统基础知识</b>	<b>88</b>
2.1 内容提要	88
2.1.1 操作系统内核与处理机管理	89
2.1.2 存储管理	91
2.1.3 文件管理	92
2.1.4 设备管理	94
2.1.5 作业管理与用户界面	95
2.2 例题分析	98
2.2.1 操作系统内核与处理机管理	98
2.2.2 存储管理	112

2.2.3	文件管理	120
2.2.4	设备管理	124
2.2.5	作业管理	127
2.3	思考练习题及答案	131
	思考练习题	131
	思考练习题答案	134
第3章	数据库基础知识	136
3.1	内容提要	136
3.1.1	数据库管理系统的功能和特征	136
3.1.2	数据库管理技术的发展	136
3.1.3	数据描述	138
3.1.4	数据模型	139
3.1.5	数据库系统的结构	140
3.1.6	关系模型和关系运算	142
3.1.7	关系数据库 SQL 语言简介	145
3.1.8	数据库设计过程	149
3.2	例题分析	149
3.3	思考练习题及答案	162
	思考练习题	162
	思考练习题答案	170
第4章	多媒体基础知识	172
4.1	内容提要	172
4.1.1	多媒体的概念	172
4.1.2	多媒体计算技术	173
4.1.3	多媒体存储技术	175
4.1.4	多媒体网络应用	177
4.1.5	多媒体内容编辑语言	180
4.2	例题分析	180
4.2.1	多媒体的概念	180
4.2.2	多媒体计算技术	181
4.2.3	多媒体存储技术	200
4.2.4	多媒体网络应用	202
4.2.5	多媒体内容编辑语言	203



4.3 思考练习题及答案	204
思考练习题	204
思考练习题答案	210
<b>第 5 章 网络基础知识</b>	<b>211</b>
5.1 内容提要	211
5.1.1 计算机网络的基本概念	211
5.1.2 计算机网络的体系结构	212
5.1.3 网络的传输控制	212
5.1.4 网络互连设备	213
5.1.5 局域网技术	214
5.1.6 广域网与接入技术	214
5.1.7 TCP/IP 与 Internet	215
5.1.8 客户机/服务器模式与网络计算	217
5.1.9 Windows NT 系统及管理	217
5.1.10 网络安全	218
5.2 例题分析	218
5.3 思考练习题及答案	240
思考练习题	240
思考练习题答案	245
<b>第 6 章 程序设计语言基础</b>	<b>246</b>
6.1 内容提要	246
6.1.1 程序语言基础知识	246
6.1.2 语言处理程序概述	246
6.1.3 构造编译程序基本知识	247
6.2 例题分析	247
6.2.1 程序语言基础知识	247
6.2.2 语言处理程序概述	254
6.2.3 构造编译程序基本知识	257
6.3 思考练习题及答案	276
思考练习题	276
思考练习题答案	282
<b>第 7 章 系统开发与运行</b>	<b>283</b>
7.1 内容提要	283

7.1.1	软件工程概述	283
7.1.2	系统分析与软件项目计划	283
7.1.3	需求分析	285
7.1.4	软件设计	286
7.1.5	编码	287
7.1.6	软件测试	288
7.1.7	面向对象方法	289
7.1.8	软件维护	290
7.1.9	软件管理	291
7.1.10	软件质量保证	292
7.1.11	软件开发工具与环境	294
7.2	例题分析	294
7.3	思考练习题及答案	318
	思考练习题	318
	思考练习题答案	333
<b>第 8 章</b>	<b>数据结构</b>	<b>335</b>
8.1	内容提要	335
8.1.1	线性表	335
8.1.2	栈	337
8.1.3	队列	340
8.1.4	数组	342
8.1.5	字符串	347
8.1.6	树与二叉树	349
8.1.7	图	355
8.1.8	排序	363
8.1.9	查找	369
8.2	例题分析	373
8.3	思考练习题及答案	397
<b>第 9 章</b>	<b>常用算法设计方法</b>	<b>430</b>
9.1	内容提要	430
9.1.1	迭代法	430
9.1.2	穷举搜索法	432
9.1.3	递推法	432

9.1.4	递归法	433
9.1.5	回溯法	434
9.1.6	贪婪法	435
9.1.7	分治法	435
9.1.8	动态规划法	436
9.2	例题分析	436
9.2.1	迭代法	436
9.2.2	穷举搜索法	439
9.2.3	递推法	442
9.2.4	递归法	444
9.2.5	回溯法	448
9.2.6	贪婪法	450
9.2.7	分治法	451
9.2.8	动态规划法	453
9.3	思考练习题及答案	457
	思考练习题	457
	思考练习题答案	460
第 10 章	标准化基础知识	462
10.1	内容提要	462
10.1.1	标准化的基本概念	462
10.1.2	标准化过程模式	462
10.1.3	标准的分类	464
10.1.4	标准的代号和编号	466
10.1.5	国际标准和国外先进标准	467
10.1.6	信息技术标准化	469
10.1.7	标准化组织	471
10.1.8	ISO9000 标准简介	472
10.1.9	ISO/IEC 15504 过程评估标准简介	473
10.2	例题分析	473
10.3	思考练习题及答案	475
	思考练习题	475
	思考练习题答案	475
第 11 章	知识产权基础知识	477
11.1	内容提要	477

11.1.1	知识产权的概念与特点 .....	477
11.1.2	我国保护软件知识产权的法律法规 .....	478
11.1.3	计算机软件著作权保护 .....	479
11.1.4	计算机软件商业秘密法律保护 .....	485
11.2	例题分析 .....	487
11.3	思考练习题及答案 .....	491
	思考练习题 .....	491
	思考练习题答案 .....	492
第 12 章	C/C++ 语言程序设计 .....	493
12.1	内容提要 .....	493
12.1.1	C 程序的构成 .....	493
12.1.2	变量的定义 .....	495
12.1.3	数据类型 .....	496
12.1.4	算术表达式 .....	497
12.1.5	赋值表达式 .....	498
12.1.6	++、-- 和逗号运算符 .....	498
12.1.7	三目运算符 .....	498
12.1.8	输入/输出 .....	499
12.1.9	选择结构 if .....	501
12.1.10	switch 语句 .....	502
12.1.11	标号语句和 goto 语句 .....	503
12.1.12	while 语句 .....	504
12.1.13	do-while 语句 .....	504
12.1.14	for 语句 .....	504
12.1.15	continue 和 break 语句 .....	505
12.1.16	字符型数据 .....	505
12.1.17	文件引用 .....	506
12.1.18	宏定义 .....	507
12.1.19	函数 .....	508
12.1.20	数组 .....	510
12.1.21	指针 .....	511
12.1.22	字符串 .....	513
12.1.23	函数的进一步讨论 .....	514
12.1.24	作用域和存储类型 .....	515



12.1.25	结构与联合.....	516
12.1.26	位运算.....	518
12.1.27	文件操作.....	518
12.1.28	C++简介.....	519
12.1.29	关于 C++的几个基本问题.....	520
12.1.30	类.....	521
12.1.31	函数重载.....	523
12.1.32	操作符重载.....	523
12.1.33	类的继承和派生.....	524
12.1.34	模板.....	529
12.1.35	异常处理.....	533
12.2	例题分析.....	534
12.3	思考练习题及答案.....	555
	思考练习题.....	555
	思考练习题答案.....	564

010	中国文学史	20.1.1
011	中国文学史	20.1.2
012	中国文学史	20.1.3
013	中国文学史	20.1.4
014	中国文学史	20.1.5
015	中国文学史	20.1.6
016	中国文学史	20.1.7
017	中国文学史	20.1.8
018	中国文学史	20.1.9
019	中国文学史	20.1.10
020	中国文学史	20.1.11
021	中国文学史	20.1.12
022	中国文学史	20.1.13
023	中国文学史	20.1.14
024	中国文学史	20.1.15
025	中国文学史	20.1.16
026	中国文学史	20.1.17
027	中国文学史	20.1.18
028	中国文学史	20.1.19
029	中国文学史	20.1.20
030	中国文学史	20.1.21
031	中国文学史	20.1.22
032	中国文学史	20.1.23
033	中国文学史	20.1.24
034	中国文学史	20.1.25
035	中国文学史	20.1.26
036	中国文学史	20.1.27
037	中国文学史	20.1.28
038	中国文学史	20.1.29
039	中国文学史	20.1.30
040	中国文学史	20.1.31
041	中国文学史	20.1.32
042	中国文学史	20.1.33
043	中国文学史	20.1.34
044	中国文学史	20.1.35
045	中国文学史	20.1.36
046	中国文学史	20.1.37
047	中国文学史	20.1.38
048	中国文学史	20.1.39
049	中国文学史	20.1.40
050	中国文学史	20.1.41
051	中国文学史	20.1.42
052	中国文学史	20.1.43
053	中国文学史	20.1.44
054	中国文学史	20.1.45
055	中国文学史	20.1.46
056	中国文学史	20.1.47
057	中国文学史	20.1.48
058	中国文学史	20.1.49
059	中国文学史	20.1.50
060	中国文学史	20.1.51
061	中国文学史	20.1.52
062	中国文学史	20.1.53
063	中国文学史	20.1.54
064	中国文学史	20.1.55
065	中国文学史	20.1.56
066	中国文学史	20.1.57
067	中国文学史	20.1.58
068	中国文学史	20.1.59
069	中国文学史	20.1.60
070	中国文学史	20.1.61
071	中国文学史	20.1.62
072	中国文学史	20.1.63
073	中国文学史	20.1.64
074	中国文学史	20.1.65
075	中国文学史	20.1.66
076	中国文学史	20.1.67
077	中国文学史	20.1.68
078	中国文学史	20.1.69
079	中国文学史	20.1.70
080	中国文学史	20.1.71
081	中国文学史	20.1.72
082	中国文学史	20.1.73
083	中国文学史	20.1.74
084	中国文学史	20.1.75
085	中国文学史	20.1.76
086	中国文学史	20.1.77
087	中国文学史	20.1.78
088	中国文学史	20.1.79
089	中国文学史	20.1.80
090	中国文学史	20.1.81
091	中国文学史	20.1.82
092	中国文学史	20.1.83
093	中国文学史	20.1.84
094	中国文学史	20.1.85
095	中国文学史	20.1.86
096	中国文学史	20.1.87
097	中国文学史	20.1.88
098	中国文学史	20.1.89
099	中国文学史	20.1.90
100	中国文学史	20.1.91

## 第 1 章 计算机系统基础知识

### 1.1 内容提要

本章主要包括以下内容：

- 计算机的基本组成和特性；
- 计算机中数据的表示方法，包括数制，数据编码（原码、补码、反码、移码）的概念及特性，定点数与浮点数；
- 字符与汉字，常用校验码生成原理；
- 算术运算和逻辑运算；
- 计算机基本结构和组成原理，包括中央处理器、存储器及输入输出系统；
- 指令系统，包括指令格式，寻址方式，指令的分类和功能；
- 计算机系统性能评价。

#### 1.1.1 计算机基本组成和特性

内容要点

- (1) 电子数字计算机的基本特性；
- (2) 计算机基本组成，冯·诺依曼结构模型；
- (3) 计算机系统包含计算机硬件和软件两大部分；
- (4) 计算机系统的层次结构和虚拟机的概念。

学习难点

(1) 冯·诺依曼结构计算机包括 5 大部件：存储器用来存放数据和程序；运算器完成算术逻辑运算，又称 ALU；输入设备和输出设备 I/O 实现与外部交换数据；控制器是整个机器控制中心，负责解释指令和发出执行指令时所需的各种控制命令。运算器和控制器合称中央处理器 CPU；存储器与 CPU 合称主机；I/O 及辅助存储器称为计算机的外部设备。计算机中采用二进制数进行运算的主要原因是物理上容易实现，运算方法简单，还可表示逻辑变量。

(2) 虚拟计算机：人们根据逻辑设计使用各种电子器件研制成功的计算机称为物理机器，是实实在在的硬件计算机。人们使用机器语言（二进制指令）与物理机器打交道。计算机系统包含硬件系统和软件系统。不同的用户使用不同层次的软件语言与计算机交往，可以看做与不同层次语言的虚拟计算机交往。因为这种机器实际上是不存在的，都必须通过编译程序等翻译成机器语言，才能在物理机器上运行。

(3) 计算机系统的层次结构：用户在不同层次使用不同语言与计算机打交道，均可实现程序要求，故可以把计算机看做一个多层次的系统。

第一层（核心层）是物理机器。人们使用二进制机器指令与机器交往。

第二层是操作系统级机器。操作系统用于管理计算机的软件和硬件资源。人们通过系统调用，方便有效地使用和管理计算机资源，把这个层次叫作操作系统级机器，也可叫作操作系统虚拟机。

第三层是汇编语言机器。人们使用容易记忆的符号表示的指令（汇编语言）与机器打交道。当然最终执行这些汇编语言时还需通过汇编器把汇编语言翻译成机器语言才能在物理机器上执行。可把这层看做汇编语言虚拟机。

第四层是高级语言机器。人们使用高级语言与机器打交道。运行程序时，首先通过编译程序把高级语言翻译成机器语言才能执行，这层也可叫作高级语言虚拟机。

## 1.1.2 数据表示

### 内容要点

#### (1) 数据分类

计算机中处理的数据有两类：数值数据和非数值数据。

数值数据指表示数量的数据，有正负和大小之分，在计算机中的数据以二进制的形式进行运算和存储。

非数值数据包括字符、汉字、声音和图像等，在计算机中处理前必须以某种编码形式转换成二进制数表示。

#### (2) 数制

常用的十进制数的计数法则是：表示一位数有 10 个不同的符号（0、1、2、3、4、5、6、7、8、9），相邻数位之间的关系是“逢十进一”（各位数的位权是  $10^n$ ），它所表示的数值是各位数按权展开的和，十进制数中的 10 称为该计数制的基数。

二进制数的基数为 2，表示一位数有两个符号：0 和 1，相邻数位之间进位关系为“逢二进一”，表示的数值为各位数按权展开的和。

二进制数各位的权。例如，一个含有 9 位整数 4 位小数的二进制数各位的权分别是： $2^8, 2^7, 2^6, 2^5, 2^4, 2^3, 2^2, 2^1, 2^0, 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}$ ，其中  $2^0$  为个位数。

上述数值对应表示的十进制数数值：256, 128, 64, 32, 16, 8, 4, 2, 1, 0.5, 0.25, 0.125, 0.0625，称为对应数位的位权，必须牢记。

为了方便，计算机中二进制数常用八进制、十六进制数表示，应熟记八进制数、十六进制数与二进制数的对应关系。

BCD 码：二—十进制数，每位十进制数用 4 位二进制数表示的数制，又可分为有权码、无权码两种方案。

#### (3) 二进制数与其他记数制数的转换方法



二进制数转换为十进制数是把二进制数各位按权展开求和。

十进制数转换为二进制数时分整数和小数两部分，分开进行转换然后相加。十进制整数部分采用“除2取余”法，直到商数为0，最后得到的余数是二进制数的最高位。十进制小数部分采用“乘2取整”的方法，首先得到的整数部分是转换成二进制小数的最高位，直到要求的精度。

以此类推十进制数转换成任意进制数（如 $r$ 进制）的方法，整数部分为“除 $r$ 取余”，小数部分为“乘 $r$ 取整”。

#### (4) 机器数

各种数据在计算机中表示的形式称为机器数，其特点是数的符号用0、1表示，如“0”表示正数，“1”表示负数。小数点隐含表示不占位置。机器数对应的实际数值称为该数的真值。

机器数有两种：无符号数和带符号数。

无符号数表示正数，没有负数，机器数中无符号位。

无符号整数的小数点，固定在该数最低位之后，是纯整数。

无符号小数的小数点固定在该数最高位之前，是纯小数。

8位二进制无符号整数的表数范围是8位全0到8位全1，即0到 $2^8-1=255$ 。

8位二进制无符号小数的表数范围是8位全0到8位全1，即0到 $1-2^{-8}$ 。

带符号机器数的最高位是表示正数、负数的符号位，其余为数值位。带符号整数的小数点固定在该数最低位之后，是纯整数。带符号小数的小数点固定在该数最高位（符号位）之后，最高数值位之前，是纯小数。这种表示数的方式称为定点数。

浮点数小数点的位置不是固定的，如 $N=MR^E$ ，小数点位置用阶码 $E$ 表示。 $E$ 为定点整数。

浮点数的数值部分用尾数 $M$ 表示， $M$ 为定点小数， $R$ 为阶 $E$ 的底数，在一个指定的机器中 $R$ 是固定的，在浮点数表示中不再出现。

#### (5) 定点数据编码方法

为了运算方便，带符号的机器数有不同的编码方法，称为码制。

① 原码：又称符号绝对值码。

该数最高位为符号位，正数用“0”表示，负数用“1”表示。其他位为数据位，用二进制数绝对值表示。原码与真值转换方便，但做加减运算不便，且零有+0和-0两种表示方法。

② 反码：正数的反码表示与原码相同。

负数的反码，符号位用“1”表示，数值位由其绝对值各位取反得到。

反码零也有+0，-0两种表示方法。因运算不便使用较少。

③ 补码：为了加减运算方便引入补码概念。关键思想是用加法代替减法。

正数的补码与原码表示相同。

负数的补码，符号位用“1”表示，数值位用其绝对值的补数表示（即原码各位求反，末位加1）。

补码最大优点是做加减运算方便。如  $(x+y)_H = (x)_H + (y)_H$ ， $(x-y)_H = (x)_H + (-y)_H$ 。

补码符号位参加运算，不单独处理，现代计算机中大都采用补码系统。补码另一优点是零的表示形式是惟一的，即  $(+0000)_H = (-0000)_H = 00000$ 。

补码的表数范围比原码、反码略宽。在定点小数中，补码可以表示-1。

$(-1)_H = 1.0000$ ，而原码、反码不能表示绝对值等于1的数。

④ 移码：为了比较两个整数的大小，引入移码概念。

移码与补码有类似的地方（数值部分），但符号位与补码相反，即正数的移码符号位为“1”，负数的移码符号位为“0”。或者说求一个数的移码，先求其补码再将其符号位变反即可得到。移码表数范围与补码整数的表数范围相同。

（6）非数值数据（符号数据）

英文字符编码的国际标准是 ASCII 码。用 7 位二进制数表示，可表示 128 个符号。扩展的二/十进制交换码 EBCDIC，采用 8bit 表示一个字符，可表示  $2^8=256$  个不同符号。

汉字编码有很多种方法。

常用的数字编码方式是区位码，将常用汉字分成 94 个区，每个区又分 94 位，每个汉字的区位编号用两个字节十进制数表示。

拼音码和字形码也是汉字常用输入编码方法。

汉字国标码也是数字编码，是汉字信息交换码国家标准，它与区位码一一对位，但区号位号用十六进制数表示，且第一个汉字放在  $(21)_{16}$  区  $(21)_{16}$  位。

计算机内存放汉字编码的方法与输入编码不同，通常用两个字节汉字国标码表示，为了与 ASCII 码区分，将每个字节最高位置“1”表示汉字字符，而 ASCII 码最高位为 0，低 7 位表示其编码，这种汉字编码称为汉字机内编码，简称内码。

汉字输出时通过内码找到其对应字模码（点阵字型）逐点输出点阵字形。如果一个汉字用  $16 \times 16$  点阵表示，则每个汉字要占 2 字节  $\times 16 = 32$  字节，两级汉字共 6763 个字模，占用大量存储空间。

（7）数据校验方法

计算机中的数据在传送、存储过程中可能出错，为了及时发现和纠正错误，编码中引入差错检查机制。

常用的校验编码有奇偶校验码、海明校验码、CRC 循环冗余校验码。奇偶校验是最常用的校验方法，可以发现一位错或奇位数同时出错。

学习难点

（1）码制

码制是为了运算方便提出的数值数据的编码方法。

① 4 种码制中，正数的原码、补码、反码表示都是一样的，即符号位为“0”，数值

位为其绝对值。而正数的移码表示中，虽然其数值也是其绝对值，但符号位相反，即正数的符号位为“1”，负数的符号位为“0”。

② 负数的原码、反码、补码的符号位都为“1”，但数值位表示方法是不同的。

原码的数值位为该数的绝对值。

反码的数值位为该数的绝对值每位求反。

补码的数值位为该数绝对值的补数，即其绝对值每位求反，末位加1。

负数的移码，其数值位与补码相同，即该数绝对值的补数，但符号位与补码不同（与原码反码也不同），即负数移码的符号位为“0”。

③ 4种编码中零的表示不同。

$$(+0)_{原} = 0000 \quad (-0)_{原} = 1000$$

$$(+0)_{反} = 0000 \quad (-0)_{反} = 1111$$

$$(+0)_{补} = 0000 \quad (-0)_{补} = 0000$$

$$(+0)_{移} = 1000 \quad (-0)_{移} = 1000$$

④ 4种编码表示数的范围不同，以8位二进制带符号整数x为例：

$$11111111 \leq (x)_{原} \leq 01111111, \quad -(2^7-1) \leq (x)_{原} \leq (2^7-1)$$

$$\text{即 } -127 \leq (x)_{原} \leq +127$$

$$10000000 \leq (x)_{反} \leq 01111111, \quad -(2^7-1) \leq (x)_{反} \leq (2^7-1)$$

$$\text{即 } -127 \leq (x)_{反} \leq +127$$

$$10000000 \leq (x)_{补} \leq 01111111, \quad -2^7 \leq (x)_{补} \leq (2^7-1)$$

$$\text{即 } -128 \leq (x)_{补} \leq +127, \quad \text{注意：此时}(x)_{补}\text{最小值是}-2^7 = -128, \text{而不是} -(2^7-1) = -127.$$

$$00000000 \leq (x)_{移} \leq 11111111, \quad -2^7 \leq (x)_{移} \leq (2^7-1)$$

$$\text{即 } -128 \leq (x)_{移} \leq +127$$

## (2) 浮点数

n位定点数的补码或移码可表示 $2^n$ 个数，而其原码、反码只能表示 $2^n-1$ 个数。表示数的范围小。

浮点数是小数点位置不固定的数，同时为了扩大定点数的表示范围，引入浮点数概念。任意浮点数N可用阶码E和尾数M两个部分来表示：

$$N = M \cdot R^E$$

M称为浮点数尾数，表示其数值的有效数字，是定点小数。

E称为浮点数的阶码，表示小数点的位置，是定点整数。

R是阶码的底。在浮点数表示中R是固定不变的，隐含表示，通常取 $R=2$ ，则 $N=M \times 2^E$ 。机器中只需给出E和M，即可知该数之数值。E和M都是定点数，也可分别指定其为原码、反码、补码、移码，同样可求出其表数范围。需要注意，对于浮点数N：

最大正数 M是最大正数，E是最大正数。

最小正数 M是最小正数，E是最小负数。

最大负数  $M$  是最大负数,  $E$  是最小负数。

最小负数  $M$  是最小负数,  $E$  是最大正数。

注意补码、移码的最小负数与原码、反码是不同的。计算机中一个数的浮点数有许多种表示方法。为了规范, 规定在运算结束将运算结果存到机器中时, 必须是规范化的浮点数, 即浮点数尾数的最高数值位是有效数字, 即  $1/2 \leq |M| < 1$ 。

### (3) 海明校验码

常用于发现纠正 1 位数据出错。其编码规则是在  $n$  位被校验数据位间, 插入  $k$  个校验位, 其校验位之个数满足关系  $2^k - 1 \geq n + k$ ; 校验位在海明码中的位置是固定的, 即海明码的 1、2、4、8... 位; 一个校验位可校验多个数据位, 每个校验位的取值等于其被校验数据位之和。其中被校数据位海明位号等于各校验位海明位号之和。

当某个数据位出错, 则引起有关的校验位改变, 当所有海明位均正确时, 则有关的校验值为全 0。

当某个校验位出错时, 则有关的校验值只有一位不为 0。且其编码为该出错校验位之海明位号。当某个数据位出错时, 则有关的校验值有 2 位或 3 位不为 0, 且其编码为该出错数据位之海明位号。

纠正其错时, 只要将出错位变反即可, 因此可以自动纠正 1 位错。发现多位数据出错或纠正多位出错的情况要复杂得多。

### (4) CRC 循环冗余校验码

用于发现和纠正信息传送过程中连续出现的多位错误。CRC 码是指在  $k$  位被校验数据之后拼接  $r$  位校验码, 得到  $k + r$  位编码。需设计一种算法, 使得发送方根据  $k$  位数据算出  $r$  位校验位之值, 一起传给对方; 接收方根据同一算法对  $k + r$  位数据进行校验, 即可判断传送是否出错。关键是找出这种算法。

我们把一个二进制代码看成一个多项式  $M(x)$  按  $x$  的降幂排列的多项式的系数。例如, 可将 1101 看成  $1x^3 + 1x^2 + 0x^1 + 1x^0$  的各个系数, 一个 4 位二进制代码看做一个三次多项式, 一个  $n$  位代码可看做  $n-1$  次多项式。代码左移 3 位相当于多项式乘以  $x^3$ , 如  $(x^3 + x^2 + 1)x^3 = x^6 + x^5 + x^3$ 。

一个多项式除以另一个多项式, 其商和余数也都是多项式, 余数多项式次数比除数多项式的次数少 1。对多项式系数的运算按模二运算进行, 模二运算时各位数据间没有进位关系。

为了产生 CRC 码的  $r$  位校验码, 我们选择一个生成多项式  $G(x)$ ,  $G(x)$  是一个  $(r + 1)$  次的多项式。

如果被校验的代码为  $k$  位二进制数据, 可用  $M(x) = M_{k-1}M_{k-2} \cdots M_1M_0$  来表示;  $k$  位数据后增加  $r$  位校验位, 则将  $M(x)$  乘以  $x^r$ , 左移  $r$  位, 再除以  $G(x)$ , 所得余数多项式  $R(x)$  为  $r-1$  次多项式, 余数相应代码为  $r$  位; 将该  $r$  位代码接在左移  $r$  位的  $M(x)$  后面得到  $M(x)x^r + R(x)$ , 它是  $G(x)$  的倍式, 可以被  $G(x)$  整除。其各位系数即构成  $k + r$  位 CRC 循环



冗余校验码。接收方收到这  $k+r$  位信息后用同样的生成多项式  $G(x)$  去除, 如果传送信息完全正确, 则应能除尽, 余数为 0。如果余数不为 0, 则说明传送出错, 并根据不同的余数判断哪一位出错。将该位变反, 即可纠正错误。显然生成多项式  $G(x)$  是经过严格挑选的, 它具有以下性质:

- 任何一位出错, CRC 码除以  $G(x)$  的余数不为 0;
- 不同的码位出错余数不能相同;
- 给定生成多项式后, 余数与出错码位之间对应关系不变, 与被校验数据无关;
- 对余数继续做模二除法, 应使余数循环。

### 1.1.3 算术运算和逻辑运算

#### 内容要点

(1) 二进制定点加减法运算是计算机算术运算的基础。

一般采用补码加减法实现, 对于定点小数:

加法  $(x+y)_n = (x)_n + (y)_n \bmod 2$

减法  $(x-y)_n = (x)_n + (-y)_n \bmod 2$

定点数的模数可看做最高位(符号位)之进位, 定点小数之模为 2。

定点数相加减, 可能出现溢出, 它不是最高位之进位。

溢出判断方法有两种: 双符号位法和进位判别法。双符号位法, 数的符号位用两位二进制数表示: 00 为正数, 11 为负数, 结果的两个符号不相同为溢出。

进位判别法判断结果溢出, 要求符号位进位  $C_n$  与次高位(最高数据位)进位  $C_{n-1}$  之中有一个有进位, 但不能同时有进位, 即  $C_n \oplus C_{n-1} = 1$ 。

(2) 定点数乘除法用原码方便, 结果的符号为运算两数符号之异或, 积(或商)为两数绝对值之积(或商)。

(3) 浮点数加减法运算步骤:

- ① 对阶, 参加运算两数阶码必须相同。
- ② 尾数加减。
- ③ 尾数规格化。
- ④ 舍入。
- ⑤ 溢出处理, 判断阶码是否溢出。

(4) 浮点数的乘除运算: 浮点数相乘, 其积的阶码为两数阶码之和, 积的尾数为两尾数之积; 浮点数相除, 其商的阶码为两数阶码之差, 商的尾数为两尾数之商; 其结果均需规格化。

(5) 关于逻辑代数与基本逻辑运算应注意以下两点:

- 必须熟记基本逻辑运算常用公式, 对逻辑表达式化简很有用。
- 逻辑运算(与、或、非、异或)都是对应位之间的运算, 相邻数位没有进位

关系。

### 学习难点

#### (1) 浮点加减法运算

① 第一步必须完成对阶操作。即两数阶码不同时，要把两数的阶码都变成大阶；此时阶码小的数要变大了，为了与原数保持相等，必须把该数之尾数右移两数阶差那么多位。

② 浮点数两尾数相加减后必须对结果进行规格化，尾数为定点小数，所谓规格化即要求尾数的绝对值要大于等于 0.5 小于 1。

在补码表示方法中，要求尾数的符号位与数值最高位不同。

即  $0.1 \times \cdots \times$  或  $1.0 \times \cdots \times$

尾数规格化有向左规格化与向右规格化之分。

当该尾数绝对值小于 0.5 时，需要左规，每左规一位，其阶码要减 1 才能保持与左规前的数相等。当尾数溢出时，需要右规。判断尾数之和是否溢出，可用双符号位法。00 表示正数符号，11 表示负数符号。当溢出时两个符号位不同，结果大于 1 为上溢，结果符号为 01；结果小于 -1 为下溢，结果符号为 10。当该尾数的绝对值大于等于 1 时，需要右规，右规一位，其阶码要加 1。

注意：此时浮点数阶的底数（基数）等于 2。

#### ③ 溢出处理。

浮点数尾数加减运算及规格化完成后，需要判断该浮点数结果是否溢出。

溢出指浮点数结果的阶码是否超过阶码能表示的范围。

阶码符号采用双符号位表示时，正数的符号为 00，负数的符号为 11；正溢时符号位为 01，负溢时符号位为 10。即阶码两个符号位不同时为溢出。正溢时，置溢出标志转溢出处理，负溢时，把结果当零看待，各位均置 0，称为机器零。

#### (2) 逻辑表达式化简

常用公式要记清：

$$A+A=A, AA=A, A+\bar{A}=1, A\bar{A}=0$$

$$A+0=A, A+1=1, \bar{\bar{A}}=A$$

$$A\oplus B=\bar{A}B+A\bar{B}, A(B+C)=AB+AC$$

$$A+(BC)=(A+B)(A+C)$$

$$\overline{AB}=\bar{A}+\bar{B}, \overline{A+B}=\bar{A}\bar{B}$$

$$A+\bar{A}B=A+B$$

证明这些公式或化简逻辑表达式可用真值表法、公式法或卡诺图法等。

如证明： $A+\bar{A}B=A+B$

$$A+\bar{A}B=A(B+\bar{A})+\bar{A}B$$

$$\begin{aligned} &= AB + A\bar{B} + \bar{A}B \\ &= AB + A\bar{B} + \bar{A}B + AB \\ &= A(B + \bar{B}) + B(A + \bar{A}) \\ &= A + B \end{aligned}$$

每年考试都有关于逻辑运算和逻辑表达式化简的考题，需引起注意。

### 1.1.4 计算机组成原理

#### 内容要点

(1) 计算机由 5 大部件组成，再加上外存储器，机构越来越复杂。5 个部件都要进行互联交换数据，常用的方案是采用总线结构。总线由内部总线与外部总线组成，内部总线连接 CPU 内模块，外部总线用于连接 CPU、存储器和 I/O 设备，又称系统总线。总线的特点是分时共享，不能有两个设备同时向总线上发送数据。

系统总线包括数据总线、地址总线、控制总线，也可采用分层总线结构，满足不同部件对数据传输速率的不同要求。

(2) 常用微机总线逐渐形成工业标准，对厂家和用户扩充设备、购买备件都是很有意义的。

(3) 中央处理器包括运算器、控制器两部分。

运算器由寄存器和算术逻辑部件构成。控制器由指令计数器、指令寄存器、指令译码器、时序电路和操作控制命令生成部件构成，是整机的控制中心，发出各种控制命令，控制各个部件实现指令功能。

(4) 控制器实现有硬布线逻辑和微程序控制两种方案。

(5) 中断系统在现代计算机系统中占据非常重要的位置，对中断控制逻辑及其工作过程需要了解清楚。

(6) 主存储器用来直接与运算器交往，又叫内存，主要要求是速度快，容量大。可分为 RAM、ROM。为了方便又研制出了 PROM、EPROM、E<sup>2</sup>PROM 和闪存 FM。

(7) 辅助存储器又叫外存，不与 CPU 直接交换数据。特点是容量大、速度慢、价格便宜。可分为磁盘、磁带等磁表面存储器和光存储器。常用光盘有 CD、CD-ROM、WORM、EOD 等。

(8) I/O 系统是计算机与外界交换数据的通道。外部设备种类繁多，速度低，数据格式、控制方式也各不相同。需要一个具有控制功能的 I/O 系统与 CPU、主存协调工作。常见外部设备种类繁多，对用户使用计算机影响很大。

(9) I/O 系统经常采用的工作方式有 3 种。

- 程序控制方式：CPU 与 I/O 串行工作，速度慢，CPU 效率低。
- 程序中断方式：CPU 与 I/O 并行工作，大大提高了 CPU 工作效率。但程序现场切换是由软件实现的，速度较慢。

- DMA 方式: DMA 控制器代替 CPU 控制 I/O 与主存直接交换数据。不需经过 CPU, 不破坏 CPU 运行程序现场, 因此速度很快。

(10) I/O 接口: 是设备与 I/O 系统和 CPU 交往的桥梁。

接口包括串行接口和并行接口。

常用磁盘接口有:

- ST506 接口, 只完成磁盘信息读写放大工作。
- ESDI 接口, 除了完成信息读写放大外, 还负责信息编码解码。
- IDE 和 EIDE 是微机常用的接口标准, 数据传输宽度分别是 8bit 和 32bit, 数据传输率可达 12Mbps~18Mbps。
- SCSI 是系统级标准通用接口, 并行传送数据宽度有 8 位、16 位、32 位 3 种。广泛应用于大容量磁盘、音频设备、CD-ROM 及 PC 兼容机中。接口除完成读放、编码解码外, 还负责数据格式串/并转换等。

微机上的新型接口: USB 为通用串行总线, 是高速总线接口, 可以串行连接多个设备; PCMCIA (个人计算机通信接口适配器) 在便携机上使用, 又叫 PC 卡插槽。

### 学习难点

#### (1) 计算机中指令的执行过程

- 取指令: 程序计数器 PC 中存放下条指令地址, 执行指令时, 首先把 PC 中的地址送主存地址寄存器, 读出指令送 CPU 中指令寄存器 IR 保存。
- 解释指令: 指令译码器翻译出指令类型、操作码功能等, 确定要执行的操作。
- 读取数据: 根据指令要求到主存中取出参加操作的数据。
- 处理数据: 被运算的数据取到运算器的数据寄存器后, 按指令操作码的要求对其进行算术或逻辑运算等操作。
- 保存结果: 把运算结果送到指定寄存单元, 同时提供下条指令地址。

计算机执行一条指令的时间称为一个指令周期, 因为机器中各种指令完成功能不同, 其指令周期长短也不相同。为了便于控制将指令执行过程分为若干阶段, 每个阶段称为一个机器周期 (CPU 周期), 如取指周期、取数周期、执行周期等。每个机器周期包括若干时钟周期 (又叫节拍电位)。

控制器负责产生有关操作命令, 这些控制命令时间上有严格的先后关系, 由控制器的时序电路产生的时序信号进行控制。

#### (2) 中断系统

当计算机运行程序过程中, 随机产生突发事件, 请求 CPU 马上处理, 此时 CPU 停下当前工作, 保存现场, 转去执行处理有关事件的服务程序, 处理完毕后, 又自动恢复现场, 返回原程序继续执行, 这个过程叫作中断。中断系统可以及时处理故障, 响应机器间通信请求, 使 CPU 与 I/O 并行工作, 大大提高了 CPU 工作效率。

中断处理过程。



① 响应中断：当中断源请求中断，且未被屏蔽，CPU 允许中断（开中断），根据中断源优先级别，在当前指令完成后响应中断请求。

② 保存现场：CPU 首先关中断并保存程序断点、程序状态字和运算器中的现场信息，转入中断服务程序入口。关中断是为了在保存现场过程中 CPU 不再响应其他中断，保存现场完毕后再开中断。

③ 执行中断服务程序。

④ 恢复现场：中断服务完毕，关中断，恢复断点及中断前的现场信息，再打开中断，恢复现场过程中不允许其他中断打扰。

⑤ 中断返回。

中断源较多时，将中断源分成若干级别，规定它们同时提出中断请求时 CPU 响应请求服务的优先顺序，中断可以嵌套，当运行低级中断服务程序时出现高级中断请求，高级中断可以打断低级中断，转去为高级中断请求服务，一般规定低级中断不能打断高级中断，同级中断也不能互相打断。中断源优先级的次序是不能改变的。

为了改变中断处理顺序，在中断系统中设置了中断屏蔽寄存器。每个中断源与一位中断屏蔽位对应，当该中断源被屏蔽时，其优先级别再高，CPU 也不响应其请求，只有当该中断源请求没有被屏蔽时，才按其优先顺序响应中断。

在计算机中，电源掉电是最高级的故障，因此规定电源掉电，不允许屏蔽，称为不可屏蔽中断，其他中断请求为可屏蔽中断。

### (3) 微程序控制器和组合逻辑控制器

控制器是计算机各部件协同工作的指挥控制中心，控制程序的执行过程，向各部件发出各种操作控制命令实现指令规定的功能。控制器包括指令部件，指令计数器给出下一条指令存放地址；指令寄存器存放本条指令，直到指令功能完成；译码器负责解释指令操作码，给出指令要求完成的功能标志。时序部件给出不同操作控制命令执行先后顺序的时间信号。操作控制部件根据操作码译码器的输出、各种时序信号和有关运算过程的各种状态信号，产生不同指令、不同 CPU 周期、不同节拍所需的各种控制命令，送到有关部件，执行指令规定的功能。

根据一条指令执行的过程，可把指令分成若干周期，按照取指、译码、取数、执行等步骤的顺序完成指令各个阶段应该做的工作。关键是如何准确地产生有关的控制命令。根据采用的方案不同，有两种控制器方案：组合逻辑控制器及微程序控制器。

组合逻辑控制器又叫硬连线控制器，它采用常规逻辑电路生成需要发出的各种操作控制命令。操作控制部件的输入是操作码译码器的输出、时序信号及程序运行的状态及结果特征，它的输出是一组带有时间标志的不同指令、不同阶段、不同状态下的操作控制命令。

微程序控制器的基本思想是一条机器指令可以分解为一系列基本操作，这些基本操作称为微操作，控制微操作的控制信号叫微命令。根据指令功能的分步要求，将完成指



定功能的微命令组合在一起,形成一条微指令,由若干条微指令组成的微指令序列完成指令每步要求,这个微指令序列叫微程序。每一条机器指令都与一个微程序相对应,如加法指令有加法微程序,乘法指令有乘法微程序,所有微程序都被存放在一个高速的微程序存储器中,这个存储器叫作控制存储器。

控制存储器用只读存储器实现。执行某一机器指令时,首先要取指令,由取指微程序实现,完成从 PC 提供的指令所在地址单元中取出指令放到控制器的指令寄存器中。根据不同的操作码转入不同指令的微程序。微程序放在控制存储器中,访问控制存储器也需要设置控制存储器地址寄存器,读出的微指令放在微指令寄存器中。微指令控制字段发出各种微命令,完成规定的操作,微指令下址字段给出下条微指令地址。

本条微指令执行完毕,下条微指令地址通常由微指令计数器(顺序执行)或微指令中的下址字段指定。

组合逻辑控制器维护、修改、扩充较困难,但速度比较快,为高性能计算机和精简指令计算机所采用。微程序控制器维护、修改、调试、扩充都很方便,在各种计算机中得到广泛使用,缺点是速度慢。

#### (4) 磁盘存储器

磁盘为磁表面存储器,信息由磁头进行读写,磁盘旋转一圈磁头在盘面上留下一个圆形轨迹称为一个磁道,信息存在磁道上。一个盘面上有许多磁道形成若干同心圆。磁道间有一定的间隔,沿磁盘半径方向每英寸磁道数称为道密度 tpi。

沿磁道方向,每英寸存储的二进制信息位数,称为位密度,(单位是 bpi),尽管各个磁道长度不同,但是规定每个磁道存储的信息位数都相同。

磁道信息单位太大,读写不便,所以将磁盘每个磁道划分成若干块,称为扇区(Sector),每个扇区存放一块数据,CPU 以块为单位读写磁盘数据,即每次读写一个数据块,显然,扇区是磁盘的最小寻址单位。各个扇区间也留有空隙。

柱面,n 个磁盘盘片组成一个盘组,所有盘面上相同的磁道组成一个圆柱面,显然一个柱面上包括 n 个磁道,若每个磁盘面上有 m 个磁道,则盘组将有 m 个柱面。

每个扇区都有起始标志、扇区地址、校验信息、间隙等,因此在新盘片使用前必须将盘片按规定的格式分区,写入有关扇区的地址标志等信息才能使用,我们称之为格式化。一个磁盘格式化后能存储的数据容量小于该磁盘格式化前的总容量。

磁盘存储器格式化后的总容量为:存放数据的盘面数 $\times$ 每面磁道数 $\times$ 每道扇区数 $\times$ 每个扇区存储的数据字节数。

存取时间,磁盘磁头接到读写信号,从当时位置移动到指定位置,并完成读写的时间叫存取时间。

存取时间包括寻道时间和寻找记录区的等待时间。因为寻找不同磁道和等待不同区域用的时间不同,通常取其平均值,称为平均存取时间。

数据传输速率指磁头找到记录区后单位时间读写的字节数,它等于一个磁道上记录

的数据字节数除以磁盘旋转一周所需的时间。

### 1.1.5 指令系统

#### 内容要点

##### (1) 指令系统的基本概念

指令系统是指计算机所能完成的机器指令的全体，反映计算机的基本性能。指令系统是使用机器语言或汇编语言的用户与机器交往的界面，指令系统决定硬件设计的复杂程度。

##### (2) 指令格式

一条指令包括操作码与地址码两个部分。操作码的位数决定指令种类。

地址码的个数影响指令字的长短和执行一条指令所需的时间。地址码是操作数在计算机内存中存放的地址。一般指令中不直接给出操作数，而是给出操作数存放的地址。根据操作数的个数又可将指令分成1地址指令、2地址指令、3地址指令等。

指令处理的数据类型包括数值数据（定点、浮点、十进制数等）、字符和字符串数据、逻辑数据等，通常由操作码指定。

##### (3) 指令的种类和功能

- 数据传送指令：实现两个单元间数据的传送，包括读写内存、进栈出栈等指令。
- 算术运算指令：包括 $+$ 、 $-$ 、 $\times$ 、 $\div$ ，定点、浮点及十进制数运算，单精度和双精度运算等。
- 逻辑运算指令：包括与、或、非、异或等操作，以及各种移位指令。需要注意，算术左移指令，最高位移入进位触发器C中，最低位补0。算术右移指令，最高位符号还要保持不变。
- I/O指令：用于输入输出操作。当I/O设备与内存统一编址时，可以用访问存储器类的指令访问I/O，不再单独设立I/O指令。
- 控制指令：包括转移指令和调用/返回指令等。

##### (4) 指令寻址方式

指令中不直接给出操作数地址，而是经过地址变换才能找到操作数的有效地址，这种变换有多种方式称为寻址方式。包括直接寻址、间接寻址、变址寻址、相对寻址、寄存器寻址、基址寻址、立即数寻址等。

#### 学习难点

(1) 指令格式中一地址指令是指指令中只给出一个操作数地址。如果操作中需要两个操作数，则隐含地指出另一个操作数在累加器AC中存放，并且本指令运算结果也放在AC中。

指令格式：

OP	A
----	---

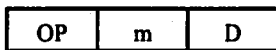
执行的操作:  $(AC) OP (A) \rightarrow AC$

一地址指令指令字长较短, 执行时只需访问内存一次取出操作数, 计算结果放在 AC 中, 不访问存储器。因此, 执行时间短, 虽然编写程序长些, 但总的效果好。这种指令要求 ALU 中只有一个累加器, 如果采用多累加器结构时, 则需指出是哪个累加器。

(2) 寻址方式: 寻找操作数有效地址的方式叫作寻址方式。

为了增加指令的灵活性, 指令中地址码部分只给出寻址方式  $m$  和位移量 (形式地址)  $D$ , 根据位移量和寻址方式可以找到操作数的有效地址  $EA$ , 不直接给出操作数地址。

指令格式:



例如一地址指令:

当  $m$  为直接寻址时, 操作数存放的有效地址  $EA=D$ 。

当  $m$  为间接寻址时, 操作数有效地址在  $D$  单元内存放, 即  $EA = (D)$ 。

当  $m$  为变址寻址时, 机器中设有一个变址寄存器  $x$ , 其内容为变址量, 操作数有效地址  $EA = (x)+D$ 。常用于修改变址量, 访问数组数据。

当  $m$  为相对寻址时, 此时指定程序计数器  $PC$  作为变址寄存器使用, 操作数有效地址  $EA = (PC)+D$ 。存放操作数的单元距本条指令的位置是固定不变的, 等于位移量  $D$ 。

当  $m$  为基址寻址方式时, 机器中有一个专用的基址寄存器  $B$ , 其内容为基地址, 操作数的有效地址  $EA = (B)+D$ 。基址寄存器为程序分配存储区或实现程序浮动而设置, 其内容一般用户不能修改。

当  $m$  为寄存器寻址方式时, 指令中地址码给出的是存放操作数的寄存器号  $R_i$ , 与直接寻址类似, 但寻找操作数不访问主存, 速度较快,  $EA=R_i$ 。

当  $m$  为寄存器间接寻址方式时, 指令地址码给出的寄存器  $R_i$  中存放的不是操作数, 而是存放操作数的有效地址  $EA$ ,  $EA = (R_i)$ 。

需要注意指令地址码中的位移量  $D$ , 可以是正数也可以是负数, 此时  $D$  是补码形式。

当  $m$  为立即数寻址方式时, 此时指令地址码部分给出的是操作数本身, 不需要再找操作数。

(3) 指令执行过程, 执行一条指令通常需要以下步骤:

- ① 根据程序计数器  $PC$  中的地址, 访问内存取出指令放入指令寄存器  $IR$  中。
- ② 根据  $IR$  中操作码字段  $OP$ , 进行指令译码, 确定指令要做什么操作。根据  $IR$  中地址码字段寻址方式  $m$ , 决定如何寻找操作数地址。
- ③ 计算存放操作数的有效地址  $EA$ 。
- ④ 根据有效地址, 访存取操作数。
- ⑤ 执行操作码  $OP$  规定的操作。
- ⑥ 保存运算结果。

⑦ 计算下条指令地址，存入程序计数器 PC 中。

有些指令不需要经过以上 7 步，根据实际需要决定执行哪几步，每执行一步叫执行一个 CPU 周期。因此执行各种指令、各种寻址方式时需要的时间是不同的，也就是说各种指令经过的 CPU 周期（机器周期）数目是不同的。但执行的前后次序是不能颠倒的。还需要指明，计算下条指令地址通常是在取指令周期中进行的，在取指周期中修改 PC 内容，这样执行一条指令时可节省一个 CPU 周期，提高了运算速度，但执行转移类等需要改变程序顺序的指令时，下条指令的地址是在本条指令结束时才送入 PC 中。

### 1.1.6 计算机系统性能评价

内容要点：

- 速度是评价计算机系统性能的重要指标
- 可靠性评价标准
- 安全性
- 病毒特性与防治

难点分析：

评价计算机系统的性能可用速度、成本、可靠性、可用性、可维护性、安全性等指标来衡量。

1. 时间是衡量计算机性能的标准，完成一定工作花费时间越少速度就越快、响应时间、吞吐率、周转时间，分别表示不同的含义。

响应时间是指在交互式系统中从输入一个命令到系统响应该命令所需的时间。

吞吐率是指计算机在单位时间内能够处理的信息数量。

周转时间是指在批处理方式下处理一个作业的平均时间，自输入设备将多个作业提交给系统，到系统完成各个作业输出处理结果所需时间与作业数的比值。

计算机完成一个作业所需的时间包括：CPU 时间、访盘时间、访存时间及 I/O 时间。但在多道程序中 CPU 等待 I/O 时，还会处理其他作业，也可减少处理一个作业所用的时间。

CPU 时间不包括 I/O 等待时间，主要指 CPU 花费在用户程序上的时间，也不包括花费在操作系统上的 CPU 时间（系统 CPU 时间）。

衡量计算机系统时间的客观标准就是机器执行程序所用的时间，如 MIPS 和 MFLOPS。

MIPS 表示每秒执行百万条指令的数目，用来描述定点运算速度，常用的有峰值 MIPS，基准程序 MIPS。 $MIPS = \text{指令条数} \div (\text{程序执行时间} \times 10^6)$

MFLOPS 表示每秒执行百万条浮点操作指令的数目，用来描述浮点指令运算速度。也可分为峰值 MFLOPS 和基准程序 MFLOPS。

考虑成本因素，有时用性能价格比作为评价系统的参数之一，它表示用每元人民币或每美元能买到多少 MIPS 或 MFLOPS。

2. 可靠性是反映产品质量的综合指标, 具有在产品使用期间的综合统计特性。

可靠性反映产品在规定条件下和规定时间内, 完成规定功能的能力, 常用平均故障间隔时间 MTBF 表示, 也可看做是产品平均无故障时间。

可维性表示产品故障时维修的难易程度, 常用平均修复时间 MTTR 表示。

可用性反映某一可维护系统某一时刻可以使用的能力, 这一指标除与产品可靠性、可维性有关外, 还与产品的管理水平有关。

计算机系统的可靠性、可用性、可维性三项指标统称为计算机的 RAS 技术。

### 3. 安全性。

计算机信息系统运行后由于天灾人祸, 很容易遭受破坏, 安全受到严重威胁, 特别是在国家最重要应用领域中如国防、金融、贸易、科学技术的应用中。计算机中所存储和处理的信息, 可能被人们窃取、篡改、破坏, 给社会和个人带来严重危害, 计算机安全和计算机犯罪问题受到社会各界的关注。

计算机安全包括计算机设备安全和数据安全。前者主要指计算机硬件设备, 包括计算机的设计、制造、运行环境、防护措施、安全管理条例等。后者主要指计算机软件(包括系统软件 and 用户软件)安全保障措施、包括人员认证(模式识别、密码技术)分级安全管理、信息隐藏及压缩技术、网络安全、入侵检测、防火墙、病毒诊断与防治等。

### 4. 计算机病毒。

计算机病毒是一种破坏性的程序, 它能够隐蔽地插入到计算机程序中, 破坏机器的功能及数据, 并且能自我复制、具有很强的传染能力。

病毒的特点是: 寄生性、隐蔽性、非法操作性、传染性、破坏性。

计算机病毒非常危险, 传播起来将引起灾难性的破坏。

防治计算机病毒是非常重要的, 除了采用技术上的措施外, 管理制度的严密性也是非常重要的。应该采取各种病毒入侵的预防措施及时检查测定病毒的出现情况, 使用各种杀病毒软件, 防止病毒的扩散和交叉感染。另外要控制软盘输入的对象和使用范围, 限制各种带毒软件的输入和运行。

## 1.2 例题分析

### 1.2.1 计算机的基本组成和特性

#### 1. 基本特性

现代计算机严格说是电子数字计算机的简称。它不同于机械计算机、电动计算机、模拟计算机, 采用的基本单元电路是高速电子开关线路, 具有非常高的运算速度, 每秒钟可完成千万次、几亿次运算。

现代计算机中被运算的数据采用二进制数表示方法, 运算规则简单, 具有非常高的



精度,而且容易实现。现代计算机的处理对象不是连续变化的电压、电流等模拟量,而是不连续变化的数字,因此又叫不连续作用计算机。

现代计算机可以进行逻辑运算,为人工智能、专家系统的研究开辟一个新天地。

现代计算机具有存储程序的能力,可以把计算过程存放在计算机中,使其能自动地、连续地执行人们预先编制的程序,为高速自动完成有关计算任务提供决定性的支持。

**【例 1-1】** 现代计算机的处理对象是\_\_\_\_。从下列各项可选答案中选择一项填入横线上。

可选答案: A. 二进制数 B. ASCII 字符 C. 十进制数 D. 电压、电流

正确答案: A

分析: 计算机可以处理文字和十进制数,但都是通过先把它们表示成二进制代码实现的。现代计算机中,处理的基本对象是二进制数据,表示二进制数的两个符号“0”和“1”是用特定的电位的高低表示的。需要注意,不是任意的电压都能识别的,也不是直接把一个连续变化的电信号送入机器中就能直接处理的。如果需要处理连续变化的电信号,也要先经过模拟信号到数字信号的转换电路(AD 转换器)变成二进制的数字信号,才能被计算机处理,因此答案选择 A。

**【例 1-2】** 存储程序的概念是冯·诺依曼 1945 年提出来的,在计算机发展史上具有特殊的意义,其重要意义是\_\_\_\_。

可选答案: A. 节省编程手续 B. 节省输入时间  
C. 提高机器处理能力 D. 保证机器自动地、连续地执行程序

正确答案: D

分析: 世界上第一台电子数字计算机 ENIAC 是 1946 年诞生的,其主要缺点是编排解题步骤困难,是通过在接线板上不同的连线方法实现的。这种做法费时费力,且容量有限。冯·诺依曼在研制过程中提出了用二进制数表示机器指令,用指令编写解题程序,按执行先后顺序依次存放在存储器中,执行程序时,依次逐条从存储器中取出要执行的指令,控制各部件完成指令规定的功能,机器中设置一个指令地址计数器,简称指令计数器或程序计数器,每取出一条指令,指令计数器自动加 1,给出下一条指令的地址,从而保证计算机可以自动地、连续地执行程序,完成程序规定的功能。因此本题正确答案选 D。

存储程序的方法,并不节省编排程序的时间,也不能节省输入步骤,主要贡献是保证机器自动地、连续地执行程序。

## 2. 计算机基本组成例题分析

计算机是现代化信息处理工具,被处理的各种信息,包括数字、字符、语音、图像等,都需要转换成二进制数据,才能在机器中进行存储、加工、传送。

计算机内的信息都是以二进制数形式表示的。每个数据字都由若干位二进制数表示,每一位二进制数称为一个 bit。一个数据字(word)包含的二进制数的位数称为字长。

不同用途的计算机，其字长是不同的，以字符为处理对象的机器字长最少是 8 位，称为一个字节 (byte)，1byte=8bit。

为了兼顾处理字节方便，计算机的字长通常定为字节的整数倍。小型机的字长多采用 16 位、32 位，中型机、大型机字长是 32 位或 64 位。按照诺依曼提出的计算机结构模型，计算机由运算器、控制器、存储器、输入设备和输出设备五大部件组成。机器运行时，利用指令编写的计算程序，按顺序存放在存储器中，机器依次执行各条指令，每取出一条指令，执行一条指令，同时给出下条指令的地址，这种控制方式，又称为控制驱动方式。

【例 1-3】计算机的基本组成包括\_\_\_\_\_。

- 可选答案：A. 运算器、控制器、存储器      B. 主机和外部设备  
C. 主机和软件系统      D. 硬件设备和操作系统

正确答案：B

分析：计算机的基本组成是指计算机的硬件组成，不包含实用程序和软件系统，运算器控制器和存储器是计算机的核心，三者合起来称为计算机的主机。计算机只有主机是无法工作的，还必须具有输入数据和程序的输入设备，及输出运算结果的输出设备才能正常工作，后二者合称外设。因此计算机的基本组成应包括主机和外部设备。这样才构成一个完整的计算机，本题正确答案选择 B。

【例 1-4】计算机的中央处理器是指\_\_\_\_\_。

- 可选答案：A. 主机      B. 运算器      C. CPU      D. 控制器

正确答案：C

分析：计算机中运算器是数据处理中心，控制器是机器的指挥控制中心，二者合称中央处理器，英文名称缩写为 CPU。本题正确答案选择 C。

计算机中主机包括 CPU 和主存两个部分，因此不应该选择 A。

【例 1-5】存储器按照一定顺序划分成许多存储单元。每个存储单元有一个编号，称为存储单元的地址。访问存储器必须按地址进行访问，存储单元内存放的是\_\_\_\_\_。

- 可选答案：A. 存储器单元的地址编号      B. 指定单元存放的数据  
C. 将要写入存储单元的内容      D. 访问存储器的控制命令

正确答案：B

分析：存储器各存储单元中存放的是该单元存放的数据。访问每一个存储单元必须先给出该单元的地址，存储单元的内容不是该单元的地址，也不是访问存储器的控制命令。将要写入存储单元的内容是先放在存储器的数据缓冲寄存器中，再在存储单元地址和写命令控制下才能写入该存储单元中。因此，正确答案选择 B。

【例 1-6】计算机存储器用来存放被运算的数据和程序，如果读出一个存储单元的内容后，该单元的内容\_\_\_\_\_。

- 可选答案：A. 清零      B. 保持不变

C. 如同小件寄存处, 该单元内容被取走

D. 不定

**正确答案: B**

**分析:** 存储器的主要功能是存放被运算的数据和程序, 只要不停电, 不写入新的内容, 其内容要保持不变。就像一个问询处, 有人来问 1010 单元住的是谁? 答: 李四。访问的人得到 1010 号住的是李四这个信息后就走了, 并没有改变 1010 号住户的姓名, 以后无论是谁来问, 只要没有搬入新的住户, 1010 号住的都是李四。也就是说读出某存储单元的内容时, 并没有改变该单元的内容。因此, 正确答案选 B。

**【例 1-7】** 总线是计算机各部件交换信息的公共通路, 当使用总线传送数据时, 在每一时刻总线上传送\_\_\_\_\_。

**可选答案:** A. 多个部件发送给多个部件的信息

B. 多个部件发送给一个部件的信息

C. 一个部件发送给一个部件的多组信息

D. 一个部件发送给多个部件的一组信息

**正确答案: D**

**分析:** 总线的特点是各个部件间传送信息时, 采用分时共享的方式使用总线。每一时刻只能有一个部件占用总线, 向总线上发送一组信息, 但是在总线上同一时刻可以由多个部件同时接收这组信息, 因为这种情况并不影响这组总线正确地传送这组信息。

一个部件利用总线, 在同一时刻向另一个部件发送一组信息是允许的, 但同一时刻一组总线上不能同时传送多组信息。

因此正确答案选择 D。

### 3. 计算机系统层次结构例题分析

一个完整的计算机系统, 包括硬件系统和软件系统两个部分。

计算机执行程序时, 只能识别用二进制数表示的指令编写的程序, 这种程序叫做机器语言程序。机器语言程序编写、检查、输入时都很困难, 容易混淆。于是人们开始研制了汇编语言和各种高级语言, 大大方便用户使用计算机, 这些语言处理程序就是程序系统的一部分。

不同的用户使用不同的语言编写的程序在机器上运行, 并得到计算结果。对于用户来说好像他们使用的是一台汇编语言计算机, 或高级程序语言计算机, 实际上并不存在汇编语言计算机或高级语言计算机, 而是用户通过汇编器或编译程序实现的, 这种计算机可以看做是一台不同层次的虚拟计算机。因此运行机器指令的计算机通过汇编器就可变成汇编语言虚拟计算机, 通过运行不同的编译程序就变成不同的高级语言虚拟计算机。

**【例 1-8】** 计算机可以运行用各种高级程序设计语言编写的程序, 但都必须经过变换变成最终计算机能够辨识的\_\_\_\_\_, 才能执行。

**可选答案:** A. 二进制机器语言

B. 汇编语言

C. 中间语言

D. 操作系统原语



**正确答案：A**

**分析：**计算机只能识别二进机器制指令。因此在运行各种高级语言编写的程序时，都必须通过编译程序、解释程序和各種转换程序，把各种高级语言转换成机器语言才能在机器上执行。

因此正确答案应选择 A。

一台计算机必须配备有关语言处理程序，才能变成可以运行该语言编写的程序的计算机。

## 1.2.2 数据表示方法

### 1. 记数制例题分析

十进制计数制，表示一位数用十个不同的符号，相邻数位间进位关系是逢十进一。

二进制计数制，表示一位数用二个不同的符号（0，1），相邻数位间进行关系是逢二进一。

**【例 1-9】** 求二进制数 1001.11 表示的十进制数是多少？

**解：**设一个二进制数

$$\begin{aligned} N &= K_3K_2K_1K_0 \cdot K_{-1}K_{-2} \\ &= K_3 \times 2^3 + K_2 \times 2^2 + K_1 \times 2^1 + K_0 \times 2^0 + K_{-1} \times 2^{-1} + K_{-2} \times 2^{-2} \end{aligned}$$

其中  $K_i$  分别为二进制数列中某个数位取值， $2^i$  为数位  $K_i$  对应的位权。

$$\begin{aligned} \text{因此} \quad (1001.11)_2 &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \\ &= (8 + 0 + 0 + 1 + 0.5 + 0.25)_{10} \\ &= (9.75)_{10} \end{aligned}$$

**【例 1-10】** 求八进制数 123.4 表示的十进制数是多少？

**解：**八进制数每个数位用 8 个不同符号表示，即 0、1、2、3、4、5、6、7，相邻数位间进位关系是逢八进一。

设一个八进制数

$$N = K_3 \times 8^3 + K_2 \times 8^2 + K_1 \times 8^1 + K_0 \times 8^0 + K_{-1} \times 8^{-1} + K_{-2} \times 8^{-2}$$

其中  $K_i$  表示八进制数列中对应位的取值， $8^i$  表示  $K_i$  对应的位权。

$$\begin{aligned} \text{因此：} \quad (123.4)_8 &= 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 + 4 \times 8^{-1} \\ &= (1 \times 64 + 2 \times 8 + 3 \times 1 + 4 \times 8^{-1})_{10} \\ &= (64 + 16 + 3 + 0.5)_{10} \\ &= (83.5)_{10} \end{aligned}$$

**注意：** $K_0$  表示数列中整数位个位，其位权取值为  $R^0=1$ 。

$R$  称为计数制的基数，二进制计数制中  $R=2$ ，八进制计数制中  $R=8$ 。

**【例 1-11】** 求十六进制数  $(123.4)_{16}$  对应的十进制数是多少？

**解：**十六进制数，每一位数可用 16 个不同的符号中的一个来表示，16 个符号分别

是 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。

相邻位间进位关系是逢 16 进一。

设一个十六进制数

$$N = K_3 \times 16^3 + K_2 \times 16^2 + K_1 \times 16^1 + K_0 \times 16^0 + K_{-1} \times 16^{-1}$$

其中  $K_i$  为十六进制数列中对应位的取值,  $16^i$  为  $K_i$  对应的位权。

$$\begin{aligned} \text{因此: } (123.4)_{16} &= 1 \times 16^2 + 2 \times 16^1 + 3 \times 16^0 + 4 \times 16^{-1} \\ &= (1 \times 256 + 2 \times 16 + 3 \times 1 + 4 \times \frac{1}{16})_{10} \\ &= (256 + 32 + 3 + 0.25)_{10} \\ &= (288.25)_{10} \end{aligned}$$

## 2. 不同数制间数据的转换例题分析

**【例 1-12】** 把十进制整数 19 转换成二进制整数。

解: 十进制整数转换成二进制整数的方法用“除 2 取余”法, 最先得到的是二进制整数的低位。另外要求除法必须做到商为“0”才停止, 而不是余数为 0。

$$\begin{array}{rll} 2 \overline{) 19} & & \\ 2 \overline{) 9} & \cdots \cdots & \text{余数 } 1 = k_0 \quad \text{低位} \\ 2 \overline{) 4} & \cdots \cdots & \text{余数 } 1 = k_1 \\ 2 \overline{) 2} & \cdots \cdots & \text{余数 } 0 = k_2 \\ 2 \overline{) 1} & \cdots \cdots & \text{余数 } 0 = k_3 \\ 0 & \cdots \cdots & \text{余数 } 1 = k_4 \quad \text{高位} \end{array}$$

因此  $(19)_{10} = (10011)_2$

**【例 1-13】** 求十进制小数 0.375 表示的二进制小数是什么?

解: 十进制小数转换成二进制小数用“乘 2 取整”法, 最先得的是二进制小数最高位。

$$\begin{array}{rll} 0.375 & & \\ \times 2 & & \\ \hline 0.750 & \text{整数部分 } 0 = K_{-1} & \text{高位} \\ \times 2 & & \\ \hline 1.500 & \text{整数部分 } 1 = K_{-2} & \\ \times 2 & & \\ \hline 1.000 & \text{整数部分 } 1 = K_{-3} & \text{低位} \end{array}$$

所以  $(0.375)_{10} = (0.011)_2$

**【例 1-14】** 求十进制数 19.375 表示的二进制数是多少?

解: 如果一个十进制数包含有整数部分和小数部分, 则将整数部分及小数部分分别求出其对应的二进制整数和二进制小数, 再将二者用小数点连接起来即是答案。

$$\begin{aligned} \text{因为 } (19)_{10} &= (10011)_2 \\ (0.375)_{10} &= (0.011)_2 \end{aligned}$$



所以  $(19.375)_{10} = (10011.011)_2$

**【例 1-15】** 将二进制数 1011.11 转换成八进制数。

解：八进制数中使用八个不同的符号，表示八个不同的符号在计算机中是很困难的，因为一个具有 8 个不同稳定状态的器件是很难找的；但一位八进制数可以用 3 位二进制数来表示，还是方便的。

如： $(0)_8 = (000)_2$      $(1)_8 = (001)_2$

$(2)_8 = (010)_2$      $(3)_8 = (011)_2$

$(4)_8 = (100)_2$      $(5)_8 = (101)_2$

$(6)_8 = (110)_2$      $(7)_8 = (111)_2$

二进制数转换为八进制数的方法，从小数点开始，整数部分向左每 3 位一组，小数部分向右每 3 位一组，不足 3 位补 0，必须补足 3 位，再分别用八进制数表示即可。

本例中  $(10111.11)_2 = (101, 111.110)_2 = (57.6)_8$

**【例 1-16】** 将八进制数 76.5 转换成二进制数。

解：按照类似道理，将每位八进制数用 3 位二进制数表示出来即可。

$(76.5)_8 = (111, 110.101)_2 = (111110.101)_2$

**【例 1-17】** 将二进制数 10111.101 转换成十六进制数。

解：用类似八进制数的转换方法，从小数点开始，分别向左、向右每 4 位一组，不足 4 位补 0，补够 4 位，再用 16 进制数的符号分别表示之。

需要注意一位十六进制数需要 4 位二进制数表示。

$(0)_{16} = (0000)_2$      $(1)_{16} = (0001)_2$

$(2)_{16} = (0010)_2$      $(3)_{16} = (0011)_2$

$(4)_{16} = (0100)_2$      $(5)_{16} = (0101)_2$

$(6)_{16} = (0110)_2$      $(7)_{16} = (0111)_2$

$(8)_{16} = (1000)_2$      $(9)_{16} = (1001)_2$

$(A)_{16} = (1010)_2$      $(B)_{16} = (1011)_2$

$(C)_{16} = (1100)_2$      $(D)_{16} = (1101)_2$

$(E)_{16} = (1110)_2$      $(F)_{16} = (1111)_2$

其中：A、B、C、D、E、F，分别对应十进制数 10、11、12、13、14、15。

本例中  $(10111.101)_2 = (0101, 1110.1010)_2 = (5E \cdot A)_{16}$

类似道理，将十六进制数转换成二进制数时可将每位十六进制数分别用 4 位二进制数表示即可。

需要注意：在计算机内为了区别十六进制，十进制、八进制、二进制数，常常在数列末尾加一个后缀来表示。

如  $(5E \cdot C)_{16}$  写成  $5E \cdot CH$  表示 H 前边的数是十六进制的数。同理用 O 表示八进制数，用 D 表示十进制数，用 B 表示二进制数。

### 3. 十进制数据编码例题分析

因为找到一个具有十个不同稳定状态的器件很难,在计算机中可以采用4位二进制数表示一位十进制数。但是4位二进制数有 $2^4=16$ 种编码,我们只从其中选用10种编码,其余6种不用。

十进制数编码有两种方案,即有权码方案和无权码方案。

有权码方案中,其4位二进制数每一位数都对应固定的权,用4位二进制数表示的和代表对应十进制数。

最常用的有权码方案是8421码,8421码每位二进制数的权与二进制计数制中每位二进制数的权相同,又可称为二进制编码的十进制数编码,简称BCD编码。

最常用的无权码是余3码和格雷码。

**【例1-18】**将二进制数1110111用十进制数BCD码表示。

解:先将二进制数按权展开求和,表示成十进制数。

$$\begin{aligned}(1110111)_2 &= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= (64 + 32 + 16 + 0 + 4 + 2 + 1)_{10} \\ &= (119)_{10}\end{aligned}$$

再将十进制数用BCD码表示。

此时把每位十进制数用其对应的BCD码表示,再把各位连起来即可。

$$(119)_{10} = (0001, 0001, 1001)_{\text{BCD}}$$

注意:BCD码中每4位二进制数表示1位十进制数,而各位十进制数间仍沿用逢十进一的十进制数进位关系。

**【例1-19】**将十进制数119用余3码表示。

解:余3码是一种无权码,表示一位十进制数的各位二进制数没有固定的位权。

余3码是在8421码基础上每位十进制数BCD码再加上二进制数0011得到的。

余3码和十进制数、BCD码间的对应关系如下所示。

十进制数	BCD 码	余 3 码
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

其他 6 个二进制表示的编码为非法码。

本例 $(119)_{10} = (0100, 0100, 1100)_{\text{8421}}$

余 3 码的主要特点是在进行十进制加减法时,各十进制数位间产生正确的进位关系。

#### 4. 字符编码例题分析

国际上普遍采用美国国家信息交换标准码 ASCII 表示英文大小写字母、阿拉伯数字、通用运算符 +、-、×、÷ 和标点符号等 128 个符号,用 7 位二进制数编码表示,在计算机中占用 1 个字节(Byte),字节最高位用“0”表示。

ASCII 码 7 位二进制数共分 8 组,每组 16 个符号。

000 组 001 组是控制字符;共 32 个,不能显示。

011 组前十个编码表示十个阿拉伯数字 0~9,按数字大小次序编码存放。

100 组 101 组表示大写英文字母,按字母顺序存放。

110 组 111 组表示小写英文字母,按字母顺序存放。

如英文字母“A”的 ASCII 编码是 01000001;阿拉伯数字“0”的 ASCII 编码是 00110000,阿拉伯数字“5”的 ASCII 编码是 00110101;依次类推。

【例 1-20】已知英文字母“A”的 ASCII 编码是 01000001,问英文字母“B”的 ASCII 编码是什么?“G”的 ASCII 编码是什么?

解:根据英文字母在 ASCII 编码中是按顺序存放的规则,“B”应是第 2 个字母,因此“B”的 ASCII 码是 0100010。“G”应是第 7 个字母,“G”的 ASCII 码是 01000111。

计算机中英文字母的存储传送都是用 ASCII 码表示的,每个字母占用一个字节。因为 ASCII 码的数值大小,也可反映字母的先后顺序,可以用 ASCII 码对英文字母进行排序。

#### 5. 汉字编码例题分析

【例 1-21】汉字区位码的区号位号各是什么含义?如何表示汉字?

解:汉字区位码是一种数字输入编码,没有重码。

区位码把常用汉字分成 94 区,每个区分成 94 位,区号、位号分别用两位十进制数表示。因此区号、位号四位十进制数可惟一地确定一个汉字,输入一个汉字时需要击键 4 次。

区位码共可表示  $94 \times 94 = 8836$  个汉字,每个汉字占两个字节。

关于汉字分区的规定:区位码中 1~15 区为常用符号;16~55 区为较常用的一级汉字 3000 多个,按音序排列;56~87 区为二级汉字,约 4000 个,按偏旁部首排列,88~94 区未用。

例如,第一个汉字“啊”放在 16 区 1 位,其区位码编号为 1601。

【例 1-22】说明区位码与国标码有什么差别?已知汉字区位码是 1601,求其国标码。

解:区位码是一种汉字输入码。输入码种类很多,为了使不同的计算机汉字系统之间交换汉字信息方便,我国制定了汉字信息交换码国家标准 GB2312-80,即国标码。

国标码也是一种数字编码,与区位码一一对应,国标码码长也是用两个字节表示一

个汉字, 字节最高位都是“0”, 但是对应的区号位号不用十进制数表示, 而是用十六进制数表示; 并且第一个汉字放在国标码的二字节平面中  $(21)_{16}$  区  $(21)_{16}$  位中。国标码的区号与区位码的区号相差  $(20)_{16}$ , 国标码的位号与区位码的位号相差  $(20)_{16}$ 。因此要将区位码的区号位号分别用十六进制的数表示, 再加上  $(2020)_{16}$  即可得到指定汉字的国标码。

本例中, 汉字“啊”的区位码是 1601, 把十进制的区号、位号分别用 16 进制数表示, 变成  $(1001)_{16}$ , 再加上  $(2020)_{16}$ , 即得汉字“啊”的国标码  $(3021)_{16}$ 。

**【例 1-23】** 说明汉字国标码与内码有什么区别, 已知汉字“啊”的国标码是 3021, 求其内码。

解: 汉字内码是计算机内存储、传送、处理汉字用的代码, 用两个字节表示, 与区位码、国标码一一对应, 但为了与机内 ASCII 字符相区分, 专门作出规定, 汉字每个字节最高位用“1”作标志。在字符编码中, 当字节最高位为“0”时表示英文字符; 当字节最高位为“1”时表示汉字。若把国标码二字节最高位改成“1”就变成汉字内码。

具体形成汉字内码时, 可把国标码加上十六进制数  $(8080)_{16}$ , 就可得到汉字内码。

本例中, 汉字“啊”的国标码是  $(3021)_{16}$ , 形成汉字内码时, 把字节最高位变成“1”, 即加上  $(8080)_{16}$ 。 $(3021)_{16} + (8080)_{16} = (B0A1)_{16}$

因此, “啊”字的汉字内码是 B0A1。

**【例 1-24】** 说明汉字输出码的特点, 计算  $16 \times 16$  点阵的汉字字形码表示国标码字库的最小容量。

解: 汉字内码是计算机内代表汉字的编号, 不能直接给出汉字的字形, 只有通过汉字的编号找到汉字的字形后才能通过输出设备显示出汉字, 得到汉字的字形。由于汉字字形码使用的信息量特别大, 因此机内存储、传送、处理汉字时, 都使用汉字内码, 只有必须输出汉字字形时, 才使用汉字字形码。

字形码也叫字模码, 最常使用的方法是点阵法, 就是将汉字写在网状方格媒质上, 将汉字字形分解成由若干点组成的点阵组合, 点阵中每一小方格有两个状态, 分别用“0”和“1”表示该小方格是黑还是白, 有点或是无点。这样每一个汉字字形都可对应一组二进制信息。

例如  $16 \times 16$  的汉字点阵, 就是每个汉字用 16 行, 每行 16 个小方格的点阵亮暗来表示的, 每个汉字需要  $16 \times 16 = 256$  个二进制位信息表示, 或者用 2 字节  $\times 16 = 32$  字节来表示。

汉字字形码放在固定的汉字字库中, 需要输出汉字时, 根据其内码可以从汉字字库中, 找出其对应的字形码, 逐行逐点地输出点阵信息, 显示出汉字字形。

本例中汉字字模码用  $16 \times 16$  的点阵表示, 即表示一个汉字字形需要 32 字节的信息, 按照 GB2312-80 规定的国标码约有 7500 个汉字, 则汉字字库共需  $32B \times 7500 = 240KB$  的容量。

## 6. 机器数编码例题分析

**【例 1-25】** 已知 5 位二进制定点整数的机器码是 11111。则: 其为原码时表示的十

进制数真值是 A；其为补码时表示的十进制数真值是 B；其为反码时表示的十进制数真值是 C；其为移码时表示的十进制数真值是 D。

5 位二进制定点整数的补码表示的最大正数是 E；表示的最小负数是 F。

- A、B: ① +1      ② -1      ③ +15      ④ -15  
 C: ① +1      ② -1      ③ +0      ④ -0  
 D: ① +1      DW② -1      ③ -0      ④ +15  
 E: ① 11111      ② 01111      ③ 10000      ④ 10001  
 F: ① 11111      ② 01111      ③ 10000      ④ 10001

正确答案: A: ④    B: ②    C: ④    D: ④    E: ②    F: ③

分析: 5 位定点二进制整数的机器码是 11111。

当 11111 为原码时, 按原码定义, 最高位为符号位, 其余位是数值位, 则其值是 -1111, 化成十进制数是  $(-15)_{10}$ 。

当 11111 为补码时, 按补码定义, 最高位是符号位, 所以是负数, 按照求补的原理, 负数的数值位各位求反末位加 1, 还原时也是数值位各位求反末位加 1。可得到其真值, 即 -0001, 所以该数的十进制数是 -1。

当 11111 为反码时, 按反码定义, 最高位为符号位, 是负数, 数值位各位求得到, 其真值为 -0000, 就是十进制数 -0, 也就是 0。

当 11111 为移码时, 根据移码定义, 最高位为符号位, 说明这个数是正数, 其数值位就是真值的绝对值即 +1111, 化成十进制数就是 +15。

5 位二进制定点整数的补码最大正数是 01111, 即  $(+15)_{10}$ ; 最小负数是 10000, 即  $(-16)_{10}$ 。

因为补码时  $(-15)_{10}$  的机器数是 10001, 并不是最小值, 还有一个比它还小的数是 10000, 这个数不是  $(-0)_{10}$ , 因为  $(-0)_{10}$  的补码是 00000, 这个数是  $(-16)_{10}$ 。

【例 1-26】数值数据在机器中可采用原码、反码、补码或移码 (又称增码) 来表示。若  $n$  位机器码用来存入定点数, 则在 A 表示方式中真值 0 的机器码是惟一的且为全“0”; 在 B 表示方式中最高位为“0”表示负号, 而为“1”表示正号; 采用反码、补码来表示小数点固定在符号位与最高有效位之间的定点数时, 可表示的真值  $X$  的范围分别为 C、D。

- A、B: ① 原码      ② 移码  
       ③ 反码      ④ 补码  
 C、D: ①  $-(1-2^{-(n-1)}) \leq X \leq (1-2^{-(n-1)})$       ②  $-(1-2^{-(n-1)}) \leq X \leq 1$   
       ③  $-1 \leq X \leq (1-2^{-(n-1)})$       ④  $-1 \leq X \leq 1$

正确答案: A: ④    B: ②    C: ①    D: ③

分析: 计算机中被运算的数值数据, 可分为定点小数、定点整数、浮点数, 它们都是采用二进制数表示的, 为了进行算术运算方便, 可采用不同的码制来表示, 常用码制有 4 种, 题中涉及了关于定点小数的 4 种编码定义、特点及数据表示范围。



(1) 原码又称符号绝对值法。它是用该数的绝对值前面加上符号“0”、“1”来表示，“0”表示正数，“1”表示负数。数码最大宽度为5位的定点小数，最大正数为0.1111，最小负数为1.1111，但 $(+0)_{\text{原}} = 0.0000$ ， $(-0)_{\text{原}} = 1.0000$ 。

需要特别注意零有两种表示方法。

(2) 补码

$$(x)_{\text{补}} = \begin{cases} x & 0 \leq x < 1 \\ 2+x & -1 \leq x \leq 0 \text{ (模2)} \end{cases}$$

即正数的补码是其本身（最高位为符号位“0”），负数的补码是用模减去该数绝对值。

例如：最大正数为 $(x_1)_{\text{补}} = 0.1111$ ，最小负数为

$$(x_2)_{\text{补}} = 2+(-1) = 1.0000,$$

$$(+0)_{\text{补}} = 0.0000, (-0)_{\text{补}} = 2+0 = 0.0000.$$

因此补码中零的表示是惟一的，且全为0。数的表示范围： $-1 \leq (x)_{\text{补}} \leq 1-2^{-(n-1)}$ 。

(3) 反码

$$(x)_{\text{反}} = \begin{cases} x & 0 \leq x < 1 \\ (2-2^{-(n-1)} + x) & -1 \leq x \leq 0 \end{cases}$$

即正数的反码是数值本身，前边符号位用0表示，负数的反码是各位数值位取反，符号位用1表示。 $(+0)_{\text{反}} = 0.0000$ ， $(-0)_{\text{反}} = 1.1111$ 。

其数值范围： $-1 < (x)_{\text{反}} < 1$ ，写得准确些是

$$-(1-2^{-(n-1)}) \leq (x)_{\text{反}} \leq (1-2^{-(n-1)}).$$

(4) 移码 通常用于整数比较大小，但若一定要套用到小数时可按下法分析。

根据定义： $(x)_{\text{移}} = 1+x$   $(-1 \leq x < 1)$

5位定点小数表示最大正数其移码： $(x_1)_{\text{移}} = 1+0.1111 = 1.1111$ ，最小负数为 $(x_2)_{\text{移}} = 1+(-1) = 0.0000$ ，

零为 $(+0)_{\text{移}} = 1+0 = 1.0000$ ， $(-0)_{\text{移}} = 1-0 = 1.0000$ 。

注意：移码与补码相似，但符号位相反。即正数的符号位为1，负数的符号位为0。该题关键是对几种码制的定义要清楚。

另外要注意：

①  $\pm 0$  的补码表示是一样的，且都是“0”，移码中零的表示也是惟一的，但都是1.0000。

② 原码、补码、反码的表示方式中，最高位是“0”表示正数，最高位是“1”表示负数。移码的符号位则相反，最高位是1表示正数，最高位是0表示负数。

③ 定点小数原码、反码的最小负数均大于-1，而补码和移码最小负数是-1。

【例 1-27】 假设计算机字长16位，用下列两种格式表示数据。

浮点格式：阶码 5 位二进制码（包括一位阶符），用移码表示，尾数 11 位二进制原码（包括一位尾符），规定阶码在前，尾数在后，阶的基数为 2。

定点整数：16 位二进制数补码表示，符号位在最高位。

若有一个 16 位的机器数 FF00（十六进制表示），则它表示的定点整数和浮点数的十进制真值为 A 和 B。

十进制数 -32 的定点整数和规格化浮点数的机器数分别为 C 和 D（用十六进制值表示）。

若上述最大定点整数用规格化浮点数表示，其机器码是 E。

A: ①  $2^{16}-2^8$  ②  $-(2^{16}-2^8)$  ③  $2^8$  ④  $-2^8$

B: ①  $-2^{14}$  ②  $-0.375$  ③  $-3 \times 2^{13}$  ④  $-0.25$

C~E: ① FFE0 ② FF0F ③ B600 ④ FBFF

正确答案：A: ① B: ③ C: ① D: ③ E: ④

分析：16 位机器数 FF00 为十六进制整数，用二进制表示为：

$$X = 1111111100000000$$

(1) 看做补码，定点整数最高位是符号位 1，为负数。数值位转换成原码，逐位求反，末位加 1。

$$(X)_{\text{原}} = 1000000100000000$$

它所代表的十进制真值  $X = -2^8$ ，所以 A 选择④。

(2) 看做浮点数，前 5 位是阶码，为移码，后 11 位为尾数。

$(E)_{\text{原}} = 11111$ ，把符号位变反，即得 E 的补码， $(E)_{\text{补}} = 01111$ ，为正数，十进制真值（是）+15。

后 11 位为尾数  $(M)_{\text{原}} = 11100000000$  为纯小数， $(M)_{\text{原}} = 1.11$ 。

浮点数  $X = 1.11 \times 2^{15} = -0.11 \times 2^{15} = -11 \times 2^{13} = (-3 \times 2^{13})$  十进制数，答案 B 选择③。

(3) 十进制数 -32 表示成定点整数： $-32 = -2^5$ 。

其原码形式：1000000000100000

其补码形式：111111111100000，即十六进制数 FFE0。

答案 C 选①。

(4) -32 表示成浮点数机器码，尾数必须是小数， $-32 = -2^5 = -2^{-1} \times 2^6$ 。

阶码是 6 用移码表示为 10110，尾数是 -0.1 用原码表示为 1.1000000000，浮点数机器码为：1011011000000000，写成十六进制表示为 B600，答案 D 选③。

(5) 16 位二进制数能表示的最大定点整数是  $0111111111111111 = 2^{15} - 1$ 。

浮点数尾数需用定点小数表示，原数  $= 2^{15} (0.11111111111111) = 2^{15} (1 - 2^{-15})$

所以浮点数阶码原码是 01111，移码是 11111。尾数原码是 0111111111，浮点机器码是 1111101111111111，写成十六进制为 FBFF。答案 E 选择④。

【例 1-28】有一个 5 位二进制定点小数（含 1 位符号），说明其原码表示的最大正

数、最小正数、最大负数、最小负数，原码“0”如何表示？

解：定点小数的符号位放在最高数值位之前，小数点约定放在符号位之后，最高数值位之前，隐含表示。

原码表示方法规定：正数的符号为“0”；负数的符号位为“1”；数值部分就是该数二进制数的绝对值。因此，5位二进制小数表示范围如下：

$$(+x)_{\text{最大}} = +0.1111, \quad (+0.1111)_{\text{原}} = 0.1111$$

$$(+x)_{\text{最小}} = +0.0001, \quad (+0.0001)_{\text{原}} = 0.0001$$

$$(-x)_{\text{最大}} = -0.0001, \quad (-0.0001)_{\text{原}} = 1.0001$$

$$(-x)_{\text{最小}} = -0.1111, \quad (-0.1111)_{\text{原}} = 1.1111$$

$$(+0) = +0.0000, \quad (+0.0000)_{\text{原}} = 0.0000$$

$$(-0) = -0.0000, \quad (-0.0000)_{\text{原}} = 1.0000$$

同理，可导出5位二进制定点整数（含1位符号位）的表数范围，需要注意0的原码有两种表示方法。

【例1-29】有一个5位二进制定点小数，说明其补码的最大正数，最大负数，最小正数，最小负数及补码“0”的表示方法。

解：为了简化减法运算，引入补码概念，目的是把减法变成加法。

补码的符号位的规定与原码相同，正数符号用“0”表示，负数符号用“1”表示。根据同余原理，在变补过程中引入了模的概念。补码的符号位参加运算，并且自动形成结果的符号。

定点二进制小数补码的模是2，可看做是该数符号位的进位，任意一个定点二进制小数加上模的N倍，是不会影响该定点数的数值的。其中N为正整数。

设有一个定点二进制小数x。

则x的补码 $(x)_{\#}$ 用其模2加上x的真值来表示。因此，可得到：

正数的补码是其本身；

负数的补码是 $2+x=2-|x|$ ，需做一次减法，或者用x的每个数位变反，末位加1得到。

如：若  $x = +0.1011$

$$\begin{aligned} \text{则} \quad (x)_{\#} &= 2 + 0.1011 \\ &= 10.0000 + 0.1011 \\ &= 0.1011 \end{aligned}$$

若  $x = -0.1011$

$$\begin{aligned} \text{则} \quad (x)_{\#} &= 2 + (-0.1011) \\ &= 10.0000 - 0.1011 \\ &= 1.0101 \end{aligned}$$

本例中5位定点二进制小数补码表数范围如下：

$$(+x)_{\text{最大}} = +0.1111, \quad (+0.1111)_{\#} = 0.1111$$

$$(+x)_{\text{最小}} = +0.0001, (+0.0001)_{\text{补}} = 0.0001$$

$$(-x)_{\text{最大}} = -0.0001, (-0.0001)_{\text{补}} = 11111$$

$$(-x)_{\text{最小}} = -0.1111, (-0.1111)_{\text{补}} = 1.0001 \text{ 并非最小}$$

$$(-1) = -1, (-1)_{\text{补}} = 2 + (-1) = 1.0000 \text{ 为最小}$$

$$(+0) = +0.0000, (+0.0000)_{\text{补}} = 0.0000$$

$$(-0) = -0.0000, (-0.0000)_{\text{补}} = 2 - 0 = 2 = 0.0000$$

需要注意：在求补过程中最高位是符号位，其进位等于模 2。一个数加减模的整数倍，其值不变，所以可将符号位的进位扔掉，而不影响数值大小。

另外定点小数负数补码的最小值是 -1， $(-1)_{\text{补}} = 1.0000$ ，而不是  $(-0.1111)_{\text{补}} = 1.0001$ 。

还有一个特性： $(\pm 0)_{\text{补}} = 0.0000$ ，表示的形式是一样的，这对一个数判零是很有意义的。

**【例 1-30】** 说明 5 位定点二进制整数的补码表数范围。

解：与定点小数补码表示方法类似，最高位为符号位，但小数点约定放在最低数值位之后，其补码表示为：

正数的补码是其本身，符号位为 0。

负数的补码：数值位是每位变反末位加 1，符号位为 1，也可用其模数加其真值求得。

需注意：整数补码的模数与小数的模数不同，不是 2，可理解为符号位之进位。对于 5 位定点二进制整数（含一位符号位），其模数是最高位之进位  $2^5$ 。

本例中，5 位定点二进制整数（含 1 位符号位），其表数范围：

$$(+x)_{\text{最大}} = +1111, (+1111)_{\text{补}} = 01111$$

$$(+x)_{\text{最小}} = +0001, (+0001)_{\text{补}} = 00001$$

$$(-x)_{\text{最大}} = -0001, (-0001)_{\text{补}} = 100000 - 0001 = 11111$$

$(-x)_{\text{最小}} = -1111, (-1111)_{\text{补}} = 100000 - 1111 = 10001$ ，最后这个数显然不是最小，比它还要小的负数是 -10000，其补码  $(-10000)_{\text{补}} = 100000 - 10000 = 10000$

需要注意：5 位定点二进制整数的补码（含 1 位符号），其表数范围中负数的最小值不是  $(-1111)_2$ ，即  $(-15)_{10}$ ，而是  $(-10000)_2$ ，即  $(-16)_{10}$ 。

其零的表示也只有一种形式，即

$$(+0) = +0000, (+0000)_{\text{补}} = 00000$$

$$(-0) = -0000, (-0000)_{\text{补}} = 100000 - 0000 = 00000$$

在对结果判零时还是方便的。

**【例 1-31】** 一个浮点数 N，其阶码 E 为 4 位定点二进制整数（含 1 位符号），尾数 M 为 6 位定点二进制小数（含 1 位符号），均用补码表示，求其表数范围。

解：先求出阶码表示范围：

$$(+E)_{\text{最大}} = +111, (+111)_{\text{补}} = 0111$$

$$(+E)_{\text{最小}} = +001, (+001)_{\text{补}} = 0001$$



$$(-E)_{\text{最大}} = -001, (-001)_{\text{补}} = 1111$$

$$(-E)_{\text{最小}} = -1000, (-1000)_{\text{补}} = 1000$$

再求出尾数 M 的表示范围:

$$(+M)_{\text{最大}} = +0.11111, (+0.11111)_{\text{补}} = 0.11111$$

$$(+M)_{\text{最小}} = +0.00001, (+0.00001)_{\text{补}} = 0.00001$$

$$(-M)_{\text{最大}} = -0.00001, (-0.00001)_{\text{补}} = 1.11111$$

$$(-M)_{\text{最小}} = -1.00000, (-1.00000)_{\text{补}} = 1.00000$$

因此浮点数 N 的表示范围是:

$$(+N)_{\text{最大}} = (+M)_{\text{最大}} \times 2^{(+E)_{\text{最大}}} = 0.11111 \times 2^{0111}$$

$$(+N)_{\text{最小}} = (+M)_{\text{最小}} \times 2^{(-E)_{\text{最小}}} = 0.00001 \times 2^{1000}$$

$$(-N)_{\text{最大}} = (-M)_{\text{最大}} \times 2^{(-E)_{\text{最小}}} = 1.11111 \times 2^{1000}$$

$$(-N)_{\text{最小}} = (-M)_{\text{最小}} \times 2^{(+E)_{\text{最大}}} = 1.00000 \times 2^{0111}$$

**【例 1-32】** 一个浮点数 N, 其阶码 E 为 4 位定点二进制整数 (含 1 位符号位), 用移码表示; 尾数 M 为 6 位二进制定点小数 (含 1 位符号位), 用原码表示, 求其表数范围。

解: 先求出阶码的最大数、最小数:

$$(+E)_{\text{最大}} = +111, (+111)_{\text{移}} = 1111$$

$$(+E)_{\text{最小}} = +001, (+001)_{\text{移}} = 1001$$

$$(-E)_{\text{最大}} = -001, (-001)_{\text{移}} = 0111$$

$$(-E)_{\text{最小}} = -1000, (-1000)_{\text{移}} = 0000$$

再求出其尾数的最大数、最小数。

$$(+M)_{\text{最大}} = +0.11111, (+0.11111)_{\text{原}} = 0.11111$$

$$(+M)_{\text{最小}} = +0.00001, (+0.00001)_{\text{原}} = 0.00001$$

$$(-M)_{\text{最大}} = -0.00001, (-0.00001)_{\text{原}} = 1.00001$$

$$(-M)_{\text{最小}} = -0.11111, (-0.11111)_{\text{原}} = 1.11111$$

求出该浮点数表示范围:

$$(+N)_{\text{最大}} = (+M)_{\text{最大}} \times 2^{(+E)_{\text{最大}}} = 0.11111 \times 2^{1111}$$

$$(+N)_{\text{最小}} = (+M)_{\text{最小}} \times 2^{(-E)_{\text{最小}}} = 0.00001 \times 2^{0000}$$

$$(-N)_{\text{最大}} = (-M)_{\text{最大}} \times 2^{(-E)_{\text{最小}}} = 1.00001 \times 2^{0000}$$

$$(-N)_{\text{最小}} = (-M)_{\text{最小}} \times 2^{(+E)_{\text{最大}}} = 1.11111 \times 2^{1111}$$

**【例 1-33】** 某机浮点数字长 16 位, 前 4 位为阶码 (最高位为阶符号位), 后 12 位为尾数 (含 1 位符号位)。有一个机器数为 1110001010000000 若阶码为移码, 尾数为反码, 其十进制真值是 A, 若阶码为移码, 尾数为原码, 其十进制真值是 B, 若阶码为补码, 尾数为原码, 其十进制真值是 C, 若阶码为补码, 尾数为补码, 其十进制真值是 D, D 规格化后其浮点数机器码是 E。

- 可选择答案: A. ① 10.78125      ② 20      ③ 0.125      ④ 20.78125  
 B. ① 0.78125      ② 20      ③ 10.125      ④ 20.75  
 C. ① 0.078125      ② 20.5      ③ 10.75      ④ 30.125  
 D. ① 10.078125      ② 20.75      ③ 0.125      ④ 20.75  
 E. ① 1111010100000000      ② 1101110100000000  
     ③ 1101010100000000      ④ 1111110100000000

正确答案: A. ②      B. ②      C. ①      D. ①      E. ③

分析: (1) 阶码为 4 位移码,  $(1110)_B$  的真值是  $(+6)_{10}$ , 尾数  $(0.01010000000)_B$  的真值是  $+0.01010000000$

$$A = 0.01010000000 \times 2^{+6} = (010100)_2 = (20)_{10}$$

所以 A 选择②

(2) 阶码为 4 位移码  $(1110)_B$  的真值是  $(+6)_{10}$ , 尾数为 12 位原码:

$$(0.01010000000)_B = +0.01010000000$$

$$B = 0.01010000000 \times 2^{+6} = (010100)_2 = (20)_{10}$$

所以 B 选择②

(3) 阶码为 4 位补码  $(1110)_B$  的真值是  $(-010)_2 = (-2)_{10}$ , 尾数为 12 位原码尾数  $(0.01010000000)_B = +0.01010000000$

$$C = 0.01010000000 \times 2^{-2} = (0.00010)_2 = (0.078125)_{10}$$

答案 C 选①

(4) 阶码 4 位补码  $(1110)_B$  的真值是  $(-010)_2 = (-2)_{10}$ , 尾数为 12 位补码:

$$\text{尾数 } (0.01010000000)_B = +0.01010000000$$

$$D = 0.01010000000 \times 2^{-2} = (0.000101)_2 = (0.078125)_{10}$$

答案 D 选择①

(5) 将数据 D 表示为规格化浮点数。

$$D = 0.01010000000 \times 2^{1110}, \text{规格化时尾数左移一位阶码}-1,$$

$$D = 0.101000000000 \times 2^{1101}$$

答案 E 选择③

## 7. 数据校验码例题分析

为了提高计算机工作的可靠性, 要求数据编码上采用能够发现传送错误或读出错误的方案, 最好是能够自动纠正错误的方案, 这种数据编码称为校验码或纠错码。

常用的校验码有奇偶校验码、海明码和循环冗余校验码 (CRC 码), 它们都是在被校数据中增加若干位校验位, 当被校数据中某些位出错时, 校验位也随之出错, 根据出错规律, 可以发现被校数据出错情况, 进而纠正错误。

【例 1-34】 已知被校验的数据为 6 位二进制数,  $D=101101$ , 求其海明码表示方法。

解：海明码是一种纠错码，既可发现错误，又能纠正错误的代码。

(1) 首先决定校验位之位数  $r$

当被校数据位  $k$  为 6 时，生成海明码的校验位之位数应满足  $2^r \geq k + r + 1$ ，也就是说校验码的  $2^r$  个状态中，必须能表示出被校数据中是哪一位出错，或哪一个校验位出错，当各位正确时也应该表示出来。

当  $r=3$  时， $2^3=8$ ，共有 8 个状态  $k+r+1=6+3+1=10$ ，共有 10 种情况，不能表示清楚。

当  $r=4$  时， $2^4=16$ ，共有 16 个状态，大于  $k+r+1=11$ ，因此选用  $r=4$ ，校验位至少取 4 位。

(2) 决定海明码校验位的位置

按海明码生成方法规定，海明校验位第  $i$  位应该放在  $2^{i-1}$  的海明位置上。

因此：当  $i=1$ ，校验位  $P_1$  的海明位号为  $2^{1-1}=2^0=1$ ，即放在  $H_1$  的位置上。

当  $i=2$ ，校验位  $P_2$  的海明位号为  $2^{2-1}=2^1=2$  即放在  $H_2$  的位置上。

当  $i=3$ ，校验位  $P_3$  的海明位号为  $2^{3-1}=2^2=4$  即放在  $H_4$  的位置上。

当  $i=4$ ，校验位  $P_4$  的海明位号为  $2^{4-1}=2^3=8$  即放在  $H_8$  的位置上。

(3) 决定数据位的位置

数据位  $D_6D_5D_4D_3D_2D_1$  由低到高依次插空放在其他海明位上。

$H_{10}H_9H_8H_7H_6H_5H_4H_3H_2H_1$

$D_6D_5P_4D_4D_3D_2P_3D_1P_2P_1$

即  $D_1$  放在海明位号为 3 的位置上，即  $H_3$ ，即  $D_2$  放在海明位号为 5 的位置上，即  $H_5$ ，即  $D_3$  放在海明位号为 6 的位置上，即  $H_6$ ，即  $D_4$  放在海明位号为 7 的位置上，即  $H_7$ ，即  $D_5$  放在海明位号为 9 的位置上，即  $H_9$ ，即  $D_6$  放在海明位号为 10 的位置上，即  $H_{10}$ 。

(4) 决定被校数据位由哪几位校验位进行校验

根据海明码生成方法规定：每个数据位由多个校验位进行校验，但被校数据的海明位号要等于校验该位数据的各位校验位海明位号之和。

$D_1$  的海明位号为 3，分成  $1+2$ ，由  $H_1$  之  $P_1$  和  $H_2$  之  $P_2$  校验，

$D_2$  的海明位号为 5，分成  $1+4$ ，由  $H_1$  之  $P_1$  和  $H_4$  之  $P_3$  校验，

$D_3$  的海明位号为 6，分成  $2+4$ ，由  $H_2$  之  $P_2$  和  $H_4$  之  $P_3$  校验，

$D_4$  的海明位号为 7，分成  $1+2+4$ ，由  $H_1$  之  $P_1$ ， $H_2$  之  $P_2$  和  $H_4$  之  $P_3$  校验，

$D_5$  的海明位号为 9，分成  $1+8$ ，由  $H_1$  之  $P_1$  和  $H_8$  之  $P_4$  校验，

$D_6$  的海明位号为 10，分成  $2+8$ ，由  $H_2$  之  $P_2$  和  $H_8$  之  $P_4$  校验。

(5) 决定各个校验位之值。

根据海明码生成方法规定：用偶校法形成校验位，校验位之值为各被校位数据之和。

$$P_1 = D_1 + D_2 + D_3 + D_4 + D_5,$$

$$P_2 = D_1 + D_3 + D_5 + D_6,$$

$$P_3 = D_2 + D_3 + D_4,$$

$$P_4 = D_5 + D_6.$$

本例中被校数据 101101。

即  $D_1 = 1, D_2 = 0, D_3 = 1, D_4 = 1, D_5 = 0, D_6 = 1,$

因此:

$$P_1 = D_1 + D_2 + D_4 + D_5 = 1 + 0 + 1 + 0 = 0,$$

$$P_2 = D_1 + D_3 + D_4 + D_6 = 1 + 1 + 1 + 1 = 0,$$

$$P_3 = D_2 + D_3 + D_4 = 0 + 1 + 1 = 0,$$

$$P_4 = D_5 + D_6 = 0 + 1 = 1.$$

最后得到被校验数据 101101 之海明码为: 1011100100。

(6) 海明码校验值。

用于判断传输数据过程出错情况。

规定海明码的校验值 C 为校验位和它所校验的各数据位之和。

$$C_1 = P_1 + D_1 + D_2 + D_4 + D_5,$$

$$C_2 = P_2 + D_1 + D_3 + D_4 + D_6,$$

$$C_3 = P_3 + D_2 + D_3 + D_4,$$

$$C_4 = P_4 + D_5 + D_6.$$

因为校验位等于其被校验的各位数据之和, 所以该校验位与其被校各位数据之和应为“0”。

在读出和传送正确, 没有出现错误时:  $C_4C_3C_2C_1 = 0000$ 。

如果某位出错, 则  $C_4C_3C_2C_1$  之值必定不是 0000, 且其编号就是出错数据位的海明位号。

如  $D_2$  出错, 由 1 变 0 或由 0 变 1, 则与  $D_2$  有关的海明校验值变为 1。

在本例中  $D_2$  出错影响  $C_1$  和  $C_3$ , 使  $C_1=1, C_3=1$ , 而与  $C_2、C_4$  无关,  $C_2=0, C_4=0$ 。

4 位 C 值形成的编码  $C_4C_3C_2C_1=0101$ , 表示出错数据位为海明位号为 5 的那一位即  $D_2$ 。

将  $H_5$  变反, 即可纠正错误, 因此说海明码是一种纠错码。

本例中被校数据为 101101, 形成的海明码为 1011100101, 若传送中  $H_5$  出错, 即  $H_5$  由 0 变成 1, 接收端得到一个错误码 1011110101, 经过校验电路得到海明校验值  $C_4C_3C_2C_1=0101$ , 不是 0000, 首先说明传送出错, 又说出错位是  $H_5$ , 将  $H_5$  变反即可得到正确数值 1011100101。

**【例 1-35】** 已知二进制数据  $M=100110$ , 生成多项式  $G(x) = 1011$ , 求其循环冗余校验码。

解: 如果被校验数据为 6 位, 希望得到 3 位校验位, 则生成多项式应选择 3 次多项式, 将被校验数据左移 3 位, 低位补 0 作为被除数, 用生成多项式  $G(x)$  作除数, 作模 2 除法求余数。

进行模 2 除法时, 若被除数高位为“1”, 则商“1”, 将被除数减去除数, 减法也采



用模2减法，由于模2加减法都是按位运算，各位间没有进位借位关系，模2加法与模2减法都是异或运算，都是相同的。

若被除数高位是“0”，则商“0”，减0时各位不变。每求1位商，余数减少1位，直到求得6位商，剩下3位余数为止。

具体求法如下：

$$\begin{array}{r}
 101001 \\
 1011 \overline{) 100110000} \\
 \underline{1011} \phantom{0000} \\
 0101 \phantom{0000} \\
 \underline{0000} \phantom{0000} \\
 1010 \phantom{0000} \\
 \underline{1011} \phantom{0000} \\
 0010 \phantom{0000} \\
 \underline{0000} \phantom{0000} \\
 0100 \phantom{0000} \\
 \underline{0000} \phantom{0000} \\
 1000 \phantom{0000} \\
 \underline{1011} \phantom{0000} \\
 011
 \end{array}$$

因此  $100110 \div 1011 = 101001$  得到余数 011，把余数 011 作为校验位放在被校验数据 100110 之后，形成 9 位 CRC 码，即 100110011。

利用 CRC 码进行数据传送，在接收端接收到本例中传送过来的 9 位 CRC 码后，用生成多项式  $G(x)$  去除接收到的数据，其余数应为 0，若余数不为 0，说明传送错误，且不同余数对应指出不同的被校数据位出错。

本例中若接收端接收到 100110011，与  $G(x) = 1011$  作模 2 除法。

$$\begin{array}{r}
 101001 \\
 1011 \overline{) 100110011} \\
 \underline{1011} \phantom{0000} \\
 0101 \phantom{0000} \\
 \underline{0000} \phantom{0000} \\
 1010 \phantom{0000} \\
 \underline{1011} \phantom{0000} \\
 0010 \phantom{0000} \\
 \underline{0000} \phantom{0000} \\
 0101 \phantom{0000} \\
 \underline{0000} \phantom{0000} \\
 1011 \phantom{0000} \\
 \underline{1011} \phantom{0000} \\
 000
 \end{array}$$

余数为 0，说明各位传送正确，不需纠错。

若余数不为 0，说明传送出错，是哪一位出错，可由出错位数与余数对照表找出，

把出错位求反、即可纠错。

若在接收端接收的数据是 100100011, 进行 CRC 校验时, 令其与  $G(x)$  作模 2 除法:

$$\begin{array}{r}
 101011 \\
 1011 \overline{) 100100011} \\
 \underline{1011} \phantom{0000000} \\
 0100 \phantom{0000000} \\
 \underline{0000} \phantom{0000000} \\
 1000 \phantom{0000000} \\
 \underline{1011} \phantom{0000000} \\
 0110 \phantom{0000000} \\
 \underline{0000} \phantom{0000000} \\
 1101 \phantom{0000000} \\
 \underline{1011} \phantom{0000000} \\
 1101 \phantom{0000000} \\
 \underline{1011} \phantom{0000000} \\
 110
 \end{array}$$

除法结果, 发现余数不为 0, 说明数据传送出错, 根据  $G(x)$  的校验余数与出错位的对照关系表, 可找出对应出错位是第 5 位, 把第 5 位变反, 即可纠正错误。

100100011 把出错位 (第 5 位) 求反, 变成 100110011 即可纠正传送错误。

注意: CRC 码的生成多项式不能随意指定, 必须具有若干特殊性质, 如:

- ① 任何 1 位数据错, 用生成多项式求得的新余数不等于 0。
- ② 不同数据位出错, 其余数不同, 且有一一对应关系。
- ③ 对余数继续作模 2 除法, 应使余数循环。
- ④ 如果传送正确, 用  $G(x)$  去除得的余数应该为 0。

### 1.2.3 运算方法

#### 1. 定点加减法运算例题分析

定点加法运算是计算机内最重要的运算, 减法可通过补码加法实现, 乘法和除法可以通过一系列加减法和移位实现。

并行加法器是算术逻辑部件 ALU 的核心和主体, 不管机器运算采用什么码制, 加法器结构本质上都是执行补码操作完成的。

已知两个定点数  $x$  和  $y$ , 将  $x$  和  $y$  用补码表示成  $(x)_n$ 、 $(y)_n$ , 则有:

$$(x+y)_n = (x)_n + (y)_n$$

$$(x-y)_n = (x)_n + (-y)_n$$

因此, 不管  $x$ 、 $y$  的真值是正数或是负数, 求和时只要把  $x$ 、 $y$  用补码表示, 就可直接做加法, 得到的结果就是  $x$  与  $y$  的和的补码, 而不需考虑谁大谁小、谁负谁正, 结果的符号就是和的补码的正确符号。

做减法时, 只要把减数符号改号, 求得 $(-y)_H$ 直接按加法运算, 最后得到的结果就是 $(x-y)_H$ 的补码。

做补码加减法, 其补码符号位和数值位一起参加运算, 最后得到的结果符号位就是两数和或差的正确符号位。

需要注意, 当补码加法溢出时, 结果的符号位将出现错误, 为了避免错误, 我们可采用双符号位法来解决, 即每一个数的补码符号位用两个相同的数来表示:

正数的符号用 00 来表示, 负数的符号用 11 来表示。

当结果的符号出现 01 或 10 时, 说明补码加减法时出现溢出, 01 表示正数溢出, 或叫上溢, 10 表示负向溢出, 或叫下溢。

**【例 1-36】** 已知两个定点二进制小数:  $(x)_H = 0.1001$ ,  $(y)_H = 0.0101$ , 求 $(x+y)_H = ?$

解: 执行加减法运算, 为防止溢出时出现错误的符号, 两个操作数都用双符号位表示, 不管  $x$ 、 $y$  是正数或是负数。

$$\begin{array}{r} (x)_H \quad \quad 00.1001 \\ + (y)_H \quad \quad + 00.0101 \\ \hline (x)_H + (y)_H \quad 00.1110 \\ (x)_H + (y)_H = (x+y)_H = 0.1110 \end{array}$$

两个正数相加, 结果的符号为正, 和也正确, 说明运算正常。

**【例 1-37】** 已知两个定点二进制数:  $(x)_H = 0.1101$ ,  $(y)_H = 0.1010$ , 求 $(x+y)_H = ?$

解: 两个操作数若不采用双符号位表示。

$$\begin{aligned} (x)_H &= 0.1101, (y)_H = 0.1010 \\ (x+y)_H &= (x)_H + (y)_H = 0.1101 + 0.1010 = 1.0111 \end{aligned}$$

两个正数相加, 结果为负数, 显然是错误的, 原因是两个数相加, 最高位数值位的和大于 1, 产生进位, 把符号位变成 1, 实际结果为正, 但结果大于 1, 溢出了, 属于上溢。

为避免溢出造成的错误, 用双符号位表示操作数的数符。

$$\begin{aligned} (x)_H &= 00.1101 \quad (y)_H = 00.1010 \\ (x+y)_H &= (x)_H + (y)_H = 00.1101 + 00.1010 = 01.0111 \end{aligned}$$

结果的符号为 01, 说明出现问题, 即结果为正, 但溢出了, 称为上溢。

今后执行补码加减法运算, 不管结果如何操作数必须用双符号位表示, 结果给出正确的答案。

**【例 1-38】** 已知两个定点二进制小数:  $(x)_H = 0.1101$ ,  $(y)_H = 0.1010$ , 求 $(x-y)_H = ?$

解:  $(x-y)_H = (x+(-y))_H = (x)_H + (-y)_H$

做减法时, 应先求出减数  $(-y)_H$ , 然后再作  $(x)_H + (-y)_H$ , 即可得到 $(x-y)_H$ 的结果。

$(y)_H = 0.1010$ , 求 $(-y)_H$ 时, 可将 $(y)_H$ 各位变反末位+1 得到

$$(-y)_H = 1.0101 + 0.0001 = 1.0110$$

$$\begin{aligned}(x-y)_H &= (x)_H + (-y)_H \\ &= 00.1101 + 11.0110 \\ &= 00.0011\end{aligned}$$

注意：最高符号位之进位，可以作为补码的模数扔掉，对结果没有影响。

【例 1-39】 已知两个定点二进制小数： $(x)_H = 1.1001$ ， $(y)_H = 0.1011$ ，求 $(x-y)_H = ?$

解： $(x-y)_H = (x)_H + (-y)_H$

先求 $(-y)_H = 1.0101$  是由 $(y)_H$ 各位变反末位+1 得到

$$\begin{aligned}(x)_H + (-y)_H &= 11.1001 + 11.0101 \\ &= 10.1110\end{aligned}$$

扔掉最高符号位之进位，结果为 10.1110 表示其差为负数，但溢出了，称为下溢。

## 2. 定点乘法运算例题分析

原码乘法是计算机中的基本运算。

原码又称符号—绝对值表示法，最高位为符号位，后面是数的绝对值。求两个数原码的乘积时，可分别求出乘积的符号和乘积的绝对值。乘积的符号按照同号两数相乘乘积为正，异号两数相乘乘积为负，因此乘积的符号可用参加运算的两数符号的“异或”求得。

两数的乘积就是两数绝对值的乘积，绝对值看做正数。两个正数做乘法时，需根据乘数每位之值（“0”或“1”），决定将被乘数加到部分积中、或是不加，根据每位乘数之位权，将部分积进行移位，以便错位进行相加，根据乘数之位权，决定进行几次加被乘数之操作。

【例 1-40】 已知 $(A)_H = 0.1011$ ， $(B)_H = 0.0101$  为定点二进制小数。

求： $(A)_H \times (B)_H = ?$

解：求二数乘积之关键是求二数绝对值之积。

(1) 做乘法前，需将被乘数放入被乘数寄存器 A 中，乘数放入乘数寄存器 B 中，将部分积寄存器 Z 清“0”，将乘数位权放入乘法次数计数器  $C_d$  中。

(2) 乘法步骤：

① 根据乘数寄存器末位之值  $B_n$ ，决定是否把被乘数加到部分积寄存器 Z 中。

若  $B_n = 1$ ，通过加法器做  $Z + B \rightarrow Z$ 。

若  $B_n = 0$ ，通过加法器做  $Z + 0 \rightarrow Z$ 。

加法操作均采用双符号位表示运算数据。

② 将乘数寄存器 B 右移 1 位，因为判断乘数之值的电路设在乘寄存器的末位；另外空出乘数寄存器的高位可用来存放双倍乘积之低位。

同时将部分积寄存器之内容右移 1 位，低位移入 B 之高位。主要是因为乘数各位数之位权不同，必须根据乘数之位权，实现错位相加。



③ 乘法次数计数器  $C_d - 1$ ，判断  $C_d = 0$ ?

若  $C_d \neq 0$ ，继续执行乘法下步操作，判断  $B_n = 1$ ?

若  $C_d = 0$ ，乘法结束，得两倍字长乘积。

(3) 最后求乘积符号，将两数符号位异或， $A_0 \oplus B_0 \rightarrow Z_0$

乘法结束时，得到两倍字长乘积，乘积高位在 Z 中，乘积低位在乘数寄存器 B 中，乘数消失。

具体乘法步骤：

部分积寄存器	乘数寄存器	说明
00.0000	0101	起始， $C_d = 4$ 乘数末位 $B_4 = 1$ ，+A
+ 00.1011		
00.1011	1010	Z、B 右移一位， $C_d - 1 = 3$ 乘数末位 $B_3 = 0$ ，+0
00.0101		
+ 00.0000	1101	Z、B 右移一位， $C_d - 1 = 2$ $B_2 = 1$ ，+A
00.0101		
00.0010	1111	Z、B 右移一位， $C_d - 1 = 1$ $B_1 = 0$ ，+0
+ 00.1011		
00.1101	0111	Z、B 右移一位， $C_d - 1 = 0$ 乘法结束
00.0110		
+ 00.0000		
00.0110		
00.0011		

乘积符号  $Z_0 = A_0 \oplus B_0 = 0 \oplus 0 = 0$ ，乘积为正。

$\therefore (A)_{16} \times (B)_{16} = 0.00110111$  高位积在 Z 中，低位积在 B 中。

### 3. 定点除法例题分析

【例 1-41】 已知两个定点二进制小数， $A = 0.1001$ ， $B = -0.1011$

用加减交替法求  $(A)_{16} \div (B)_{16} = ?$

解：加减交替法除法即不恢复余数除法，因为恢复余数时需多做若干次加法，使除法过程变慢，速度降低。

原码除法，符号位单独处理，看做两个正数相除。

加减交替法与恢复余数除法之区别在于：

恢复余数除法：每次减除数后，根据余数决定上商及是否恢复余数。

余数为正，上商“1”，不恢复余数，下次做减法。

余数为负，上商“0”，恢复余数，下次做减法。

加减交替法：每次根据余数决定上商和下次做加法还是做减法，但都不恢复余数。

余数为正, 上商“1”, 不恢复余数, 下次做减法。

余数为负, 上商“0”, 不恢复余数, 下次做加法。

因为不需恢复余数, 所以除法操作中, 加减法的次数固定, 速度比恢复余数除法快。

本例 $(A)_{原} = 0.1001$

$(B)_{原} = 1.1011$

做原码除法时, 不考虑数的符号, 都作为正数进行运算。因此 B 虽为负数, 但原码除法求商的绝对值时, 可把 B 当作正数看。商的符号单独处理。

+B 操作: +0.1011

-B 操作: +(-0.1011)<sub>补</sub> = 1.0101

加减交替原码除法过程如下:

被除数寄存器 A,	商寄存器 C	说明
0 0 . 1 0 0 1	0 . 0 0 0 0	起始
+ 1 1 . 0 1 0 1		-B, 判溢出
1 1 . 1 1 1 0	0 . 0 0 0 0	$R_0 < 0$ , 商“0”, 商于商寄存器末位, 不溢出
1 1 . 1 1 0 0	0 . 0 0 0 0	A, C 左移 1 位
+ 0 0 . 1 0 1 1		+B
① 0 0 . 0 1 1 1	0 . 0 0 0 0	$R_1 > 0$ , 商“1”
0 0 . 1 1 1 0	0 . 0 0 0 1	A, C 左移 1 位
+ 1 1 . 0 1 0 1		-B
① 0 0 . 0 0 1 1	0 . 0 0 0 1	$R_2 > 0$ , 商“1”
0 0 . 0 1 1 0	0 . 0 0 1 1	A, C 左移 1 位
+ 1 1 . 0 1 0 1		-B
1 1 . 1 0 1 1	0 . 0 0 1 1	$R_3 < 0$ , 商“0”,
1 1 . 0 1 1 0	0 . 0 1 1 0	A, C 左移 1 位
+ 0 0 . 1 0 1 1		+B
① 0 0 . 0 0 0 1	0 . 0 1 1 0	$R_4 > 0$ , 商“1”

被除数放在被除数寄存器 A 中, 除数放在除数寄存器 B 中, 商从商寄存器末位开始上商。

加减交替法共作 5 次加法 4 次移位, 求得 5 位商, 第 1 位商实际上是判溢出否, 结束运算。

商的符号, 二数异号  $A_0 \oplus B_0 = 0 \oplus 1 = 1$ , 商为负,

$(A)_{原} \div (B)_{原} = 0.1001 \div 1.1011 = 1.1101$

余数左移 4 次后  $R_4 = 0.0001$

真正余数  $R = 0.0001 \times 2^{-4}$

#### 4. 逻辑运算例题分析

逻辑运算处理的逻辑变量是“真与假”两个相反的逻辑概念, 用二进制数 0, 1 来表示。

常用的逻辑运算有逻辑加、逻辑乘、异或、求反等运算，可直接使用或门、与门、异或门、非门实现。

所有的逻辑运算都是按位运算，相邻各位没有进位关系。

**【例 1-42】** 如何将 8 位寄存器中的数据最高位置“1”、置“0”？如何将 ASCII 码中低 4 位分离出来？

解：利用逻辑运算可以解决这些问题。

任何一位二进制数 S，经过下列运算可得下列结果：

$$S \vee 1 = 1 \quad S \wedge 0 = 0 \quad S \oplus 1 = \bar{S}$$

因此 8 位寄存器中的数据 A，通过  $A \vee 80H$  可将最高位置“1”。

通过  $A \wedge 7FH$  可将最高位置“0”。

通过  $A \oplus 80H$  可将最高位变反。

如果需要将 ASCII 码低 4 位分离出来，可进行下列逻辑操作： $(ASCII) \wedge 0FH$

**【例 1-43】** 化简逻辑函数  $F = AB + \bar{A}C + BCD$

解：利用基本逻辑代数公式化简

$$\begin{aligned} F &= AB + \bar{A}C + BCD \\ &= AB + \bar{A}C + BC + BCD \\ &= (AB + \bar{A}C) + (BC + BCD) \\ &= AB + \bar{A}C + BC \\ &= AB + \bar{A}C \end{aligned}$$

其中用到互补律  $A + \bar{A} = 1$  0-1 律  $A + 1 = 1$

吸收律  $A + AB = A$ ， $\therefore A + AB = A(1 + B) = A$

第二吸收律  $A + \bar{A}B = A + B$

$\therefore A + \bar{A}B = A + AB + \bar{A}B = A + B(A + \bar{A}) = A + B$

$AB + \bar{A}C + BC = AB + \bar{A}C$

$$\begin{aligned} \therefore AB + \bar{A}C + BC(A + \bar{A}) &= AB + \bar{A}C + ABC + \bar{A}BC \\ &= AB + ABC + \bar{A}C + \bar{A}BC \\ &= AB + \bar{A}C \end{aligned}$$

## 5. 浮点数加减法运算例题分析

浮点数表示范围宽，运算精度高，在现代计算机中得到广泛应用。

浮点数包括阶码和尾数两个部分，阶码用来表示数据范围大小，用定点整数表示，尾数表示数据精度，用定点小数表示。阶码和尾数的功能不同，但都是定点数，定点数运算方法和定点运算装置对它们都是适用的。

**【例 1-44】** 已知两个十进制数，先把十进制数化成二进制浮点数，再按浮点数加法步骤求出二数之浮点和。

规定浮点数用 4 位阶码（含 1 位符号），5 位尾数（含 1 位符号）表示，二者均采用

补码。

求：十进制数， $0.375 + \frac{13}{16} = ?$

解：先把十进制数化成二进制数

$$A = (0.375)_{10} = (0.0110)_2 \quad B = \left(\frac{13}{16}\right)_{10} = (0.1101)_2$$

再把两个数化成浮点规格化数：

$$A = 0.0110 = 0.1100 \times 2^{-1} = 0.1100 \times 2^{1111}$$

小数左移 1 位，阶码 - 1，保持原数大小不变。

$B = 0.1101 \times 2^{0000}$  已是规格化数。

求  $A + B = ?$

① 对阶求阶差，两个浮点数阶码不等，尾数不能相加，求阶差  $\Delta E = E_A - E_B = 1111 - 0000 = 1111$

阶码为负， $E_B$  为大阶，取共同的阶码为大阶  $E_A' = E_B = 0000$

小阶之阶码变大时，为使原数大小不变，小阶的数的尾数应该右移变小才行。

$\Delta E = -1$   $E_A' = E_A + 1$ ，其尾数  $M_A$  应该右移 1 位，保持 A 的大小不变， $M_A' = 0.0110$

② 求和，两个浮点数阶码相等，求和时把两个尾数直接相加即是和的尾数，和的阶为两个数共同的阶

$$\begin{aligned} M_Z &= M_A' + M_B = 00.0110 + 00.1101 \\ &= 01.0011 \end{aligned}$$

说明尾数的和溢出。

③ 规格化，结果的尾数大于 1，属于溢出，但这种溢出可以通过调整阶码使其变成规格化的数。

采用右规的办法，把尾数右移 1 位，阶码加 1 实现， $M_Z' = 00.10011$   $E_Z' = 0000 + 0001 = 0001$

④ 舍入，规格化时，丢掉尾数最低位 1，为了减少误差，对丢掉的尾数进行舍入处理。通常采用恒置“1”法，不管丢掉的是何值，舍入时都是把尾数末位变成“1”。

因此舍入后  $M_Z' = 0.1001$

⑤ 判断溢出，判断浮点数是否溢出，就是在尾数规格化后判断其阶码是否溢出。

4 位阶码，用补码表示，最大值是 0111，最小值是 1000。现在  $E_Z' = 0001$ ，不超过阶码表示范围，所以浮点数不溢出。

最后结果其和是  $Z = 0.1001 \times 2^{0001}$

**【例 1-45】** 已知两个浮点数，原码表示。



$$x = 0.10110 \times 2^{010} \quad y = 0.11011 \times 2^{001}$$

求:  $x - y = ?$

解: 两个浮点数阶码不相等, 尾数不能直接相加减。

① 对阶, 求阶差

$$\Delta E = E_x - E_y = 010 - 001 = 001$$

阶差为正,  $E_x$  为大阶, 把  $E_y$  变成大阶, 则  $M_y$  必须右移  $\Delta E$  位, 才能使  $y$  的大小不变。

$$E_y' = E_x = 010 = E_z,$$

$M_y' = 0.011011$ , 因为阶差 = 1,  $M_y$  右移 1 位与  $y$  值相等。

② 尾数相减

$$\begin{aligned} M_z &= M_x - M_y' = 0.10110 - 0.011011 \\ &= 0.10110 + (-0.011011)_{\text{补}} \\ &= 00.10110 + 11.100101 \\ &= 00.010001 \text{ 丢掉最高符号位之进位} \\ M_z &= 0.010001 \end{aligned}$$

③ 规格化

差的尾数为正数, 0.010001 不是规格化数, 必须把最高数据位变成“1”, 才算规格化数。可把该数左移 1 位, 阶码减 1, 保持与原数大小不变, 称之为向左规格化。

$$M_z' = 0.100010 \quad E_z' = 010 - 001 = 001$$

④ 舍入

$M_z' = 0.100010$  尾数 6 位, 扔掉末位, 把数据最低位置“1”。因为末位已经是 1, 结果不变。  $M_z' = 0.10001$

⑤ 判溢出

该浮点阶码 3 位, 最大正数是 011, 最小负数是 111。现在结果的阶码是 001, 不溢出。

结果  $x - y = 0.10001 \times 2^{001}$

## 1.2.4 计算机组成原理

计算机包括五大部件: 运算器是数据处理中心, 控制器各部件工作的指挥中心, 运算器和控制器合称中央处理器 CPU, CPU 和主存储器全称主机。其他部件称为计算机的外围设备, 各部件通过总线交换数据。

### 1. 运算器例题分析

运算器是计算机数据处理中心, 主要功能是对数据进行算术运算和逻辑运算。运算器的核心是一个并行加法器, 运算结果的特征, 如溢出, 结果为“0”等需保存在专门的标志寄存器中。

运算器中设置一组寄存器, 用来存放参加运算的数据和中间结果, 区别于单累加器

结构,通用寄存器中每个寄存器都可以存放运算结果,程序员通过指令使用这组寄存器。

为了完成乘除法运算,运算器中提供左右移位功能,设置乘商寄存器。

**【例 1-46】** 运算器中各寄存器间如何交换数据? 运算器与存储器和 I/O 如何交换数据?

解:运算器的核心是算术逻辑部件 ALU,是整机的数据处理中心,计算机中各部件的数据都需要送到 ALU 中来处理。ALU 有两个输入端,送入两个被运算的数据,一个输出端送出运算结果。运算器中有多个数据寄存器通过两个多路数据选择器,选择需要的寄存器中的数据,分别送给 ALU 的两个数据输入端,经过算术逻辑部件处理,由 ALU 输出。外部数据,例如存储器和 I/O 等需要送往通用寄存器的数据,也需要经过 ALU 输入多路选择器,经过 ALU 再送入指定的寄存器中,这样可以节省寄存器间连接的线路,也比较整齐,我们把运算器中各寄存器间交换数据的公共通路叫内总线,或 CPU 总线,这种结构连线较少,扩充容易。

计算机中各部件包括存储器与 I/O 设备都需要与 ALU 连系,都需要建立与 ALU 的连接通路,因此 ALU 是整个计算机的数据传输中心。冯·诺依曼在总结现代计算机的特征时,专门指出整个计算机以运算器为中心。

**【例 1-47】** 运算器中设置标志寄存器有什么用处?

分析:运算器执行算术运算或逻辑运算,得到运算结果。这个结果往往成为后续指令程序分支的条件,因此需要保存运算结果的特征。如结果为零、结果为负、结果溢出、结果有进位等,保存运算结果特征的寄存器,称为标志寄存器,有时也叫程序状态字 PSW。每种特征在标志寄存器中用一位触发器来表示。

如运算结果为“0”时,其对应特征位  $Z = 1$ ,若  $Z = 0$ ,表示结果不为“0”;如结果溢出时,其对应特征位  $V = 1$ ,若  $V = 0$ ,表示结果不溢出,同理  $C = 1$  表示结果有进位, $S = 1$ ,表示结果为负等。

**【例 1-48】** 双端口存储器有什么特点和用处?

解:一般存储器都是给一个地址,读出一个数,属于单端口存储器,不能同时给两个地址读出两个数。为了提高访存的速度,希望有能够给两个地址,同时读出两个数的存储器,这里说的存储器当然是对一个存储器而言,不是两个存储器。如果这样,访存速度可以提高一倍,这种存储器称为双端口存储器,在这种存储器内部需要设置两套独立的读出控制电路。

注意:为了防止两个端口同时向一个存储单元写入不同数,规定写入双端口存储器时,只允许一个端口作为写入端口,但两个端口可以同时读出同一个存储单元,不会发生矛盾。

运算器中,ALU 操作,多数情况需要两个操作数,如果两个操作数都放在一个类似存储器的通用寄存器组中,读出两个操作数,需要两步才能读出。为了提高运算速度,希望 ALU 运算时,一拍能够读出两个操作数,这就要求通用寄存器组具有双端口读出

的功能,需要设置两套独立的读出控制电路,分别同时给出两个地址,从两个端口同时读出两个操作数。

这种双端口通用寄存器,给两个地址,可以同时读出两个操作数,供给 ALU 两个输入端,在一拍时间内 ALU 可给出运算结果。当两个操作数都来自同一个寄存器,也是允许的,但写入通用寄存器时,不允许同时写入两个操作数,防止同时写入同一个寄存器时造成冲突。所以只设置一套写入控制电路。在运算器中,这种设置不影响写入速度,因为 ALU 只能产生一个运算结果,保存在指定的一个寄存器中就可以了,不需要同时写入两个数据。

## 2. 控制器

控制器是整个机器的控制中心,负责程序中指令的执行顺序,发出各种控制命令控制各个部件完成指令规定的功能。也负责处理机器中出现的各种异常情况。

要求掌握指令执行过程。

掌握控制器的功能和组成原理。

了解组合逻辑控制器和微程序控制器的工作原理。

### (1) 指令执行过程例题分析

控制器负责发出各种控制命令,完成指令规定的功能,就必须了解指令的执行过程,了解控制器是如何工作的。

**【例 1-49】**说明一地址访存指令的执行过程。

解:一地址指令中只给出一个地址,根据这个地址去主存中取一个操作数,与累加器中的操作数进行操作码规定的运算,运算结果放在累加器中。

① 取指令,根据程序计数器指出的本条指令地址,把该单元的内容(指令)取出来,送到控制器的指令寄存器中保存。

② 分析指令,指令字是计算机执行具体操作的根据,必须认真分析指令字中各字段的含义,特别是操作码规定的操作,分析操作码的工作是由操作码译码器完成的。

③ 计算操作数有效地址,一般指令都不直接给出操作数的有效地址,而是按照寻址方式规定的方法寻找操作数的有效地址。

④ 访存取操作数,按照寻址方式得到操作数的有效地址后,就要到内存中把操作数取到运算器中准备进行运算。

⑤ 执行操作码规定的操作,一地址指令规定把累加器中的数与内存取来的数进行运算,结果放在累加器中。

⑥ 给出下条指令地址,保证程序可以连续不断地执行下去。通常采用将程序计数器加 1 的办法,给出下条指令地址,因为程序中指令执行顺序与指令编写和在内存中存放的顺序是一样的。这一步操作一般安排与第①步同时进行,可省一步时间。

如果指令不需要访存取数,则可省去第③④步。

一般指令至少必须包括三步:取指令、分析指令和执行指令。

## (2) 控制器的功能和组成例题分析

**【例 1-50】** 说明控制器在计算机中的地位和作用。

解：计算机是处理数据的一种机器，处理什么数据，怎么处理数据都是由控制器来控制的，因此，控制器是全机的指挥中心。

控制器具体功能包括：

- ① 决定指令执行顺序，由控制器给出下条指令在存储器中存放的地址。
- ② 分析本条指令的功能，执行什么运算，因为不同的操作发出的控制命令是不同的。
- ③ 控制指令执行过程，计算机各部件执行什么操作都是由控制器发出的控制命令来决定的。

④ 处理随机出现的各种异常状况。

**【例 1-51】** 说明指令部件的组成和功能。

解：指令部件包括指令计数器、指令寄存器、指令译码器，三个部件都直接与本条指令有关。

① 指令计数器 PC，又叫程序计数器，给出本条指令的地址，通常  $(PC) + 1$  给出下条指令地址，本条指令开始执行时，首先把  $(PC)$  送给主存地址寄存器，发读命令，即可取出本条指令。

② 指令寄存器 IR，主存取出指令后经数据总线送给控制器，将本条指令存放在指令寄存器中，在本条指令执行期间，指令寄存器的内容保持不变。

③ 指令译码器，指令取出后，必须辨认出本条指令是什么指令，以便发出不同控制命令，这一工作是由指令译码器完成的，关键是操作码译码器，把二进制的操作码翻译成相应指令的专门控制线上的控制电位，在该指令控制电位控制下发出若干专门操作控制信号，送给有关执行部件完成指令功能。

**【例 1-52】** 说明控制器的基本组成及功能。

解：① 指令部件，包括指令计数器、指令寄存器、指令译码器。给出本条指令地址，保存本条指令并加以分析。

② 时序系统，产生各种定时信号。用来决定控制信号的先后顺序。

③ 操作控制部件，产生各种操作控制命令。

④ 中断控制逻辑，处理各种随机事件。

**【例 1-53】** 说明计算机中时序电路的作用。

执行一条指令可划分成若干阶段，首先要取指令，指令未取出前，就不能分析指令，没有寻找到操作数的有效地址，就不能去取操作数，没有操作数就无法执行各种运算。每一个操作控制命令发出的时间顺序有严格规定，不能前后颠倒，为了保证操作控制命令的时间顺序。必须提供标志时间先后顺序的时序信号。

时序电路用来产生控制器需要的各种时间信号，通常包括表示取指令、分析指令、



执行指令各个阶段的机器周期时间信号, 在每个机器周期中又要分为若干节拍电位; 每个节拍中包括工作脉冲信号。

**【例 1-54】** 什么叫指令周期? 什么叫机器周期 (CPU 周期)? 什么叫节拍电位? 什么叫工作脉冲? 各有什么用途?

解: 指令周期指从取指令开始完成一条指令所需的全部时间。

按照指令执行过程划分成若干功能阶段, 每一个阶段完成一定的功能操作, 每一个阶段称为一个机器周期, 又叫 CPU 周期, 如完成取指令功能的阶段称为取指令周期, 完成分析指令的功能阶段称分析指令周期, 同理完成变址计算功能的阶段称变址周期, 完成从主存取数的功能阶段叫取数周期, 按操作码规定的要求完成规定运算的功能阶段称执行周期等。显然不同指令功能不同, 需要的机器周期数目是不同的。

在每个机器周期 (CPU 周期) 中, 控制器发出若干操作控制命令控制有关部件完成规定的功能。这些控制命令也不是同时发出的, 也是按照一定时间顺序发出的, 如取指令周期, 操作时必须先把 PC 中的指令地址送到内存的地址寄存器, 然后才能发读内存的命令, 最后才能把内存单元取出的指令送到控制器的指令寄存器, 这几步操作前后次序也是不能颠倒的。为了表示机器周期内各控制命令的先后顺序, 时序电路又提供了节拍电位信号, 用来区分每个机器周期内的时间顺序关系。通常节拍电位的宽度与时钟周期一致, 又称时钟周期。

工作脉冲。在每一个节拍期间可能要执行寄存器接收数据的工作, 因此还需提供打入脉冲, 对应某个节拍特定的时间脉冲, 称为工作脉冲。工作脉冲的宽度与时钟脉冲一致。

### (3) 处理器总线及数据通路例题分析

**【例 1-55】** 什么叫处理器总线?

解: 计算机的功能越来越强, 计算机内部的设置越来越复杂, 特别是 CPU 内部各个部件, 各个寄存器间要传送加工数据, 就需把它们通过硬件线路连接起来。为了减少连线的数目和复杂性, 通常利用算术逻辑部件 ALU 作为数据传输的枢纽, 把 ALU 的输出通过一组公用的数据线与多个寄存器相连, 这组公用数据线称为处理器总线, 又叫 CPU 总线。

**【例 1-56】** 什么叫内总线? 什么叫外总线? 什么是数据通路?

内总线指 CPU 内部各寄存器、各功能部件间之连线, 通常指 ALU 的输出到各寄存器间之公用数据线, 又叫处理器总线或 CPU 总线。

外总线指 CPU 与存储器、外部设备间之公用总线, 即系统总线, 包括数据总线、地址总线、控制总线。

总线之特点是分时共享, 向总线上发送数据时, 只允许一个部件使用总线, 但接收总线上传来的数据时, 可允许多个部件同时接收。

计算机中传送数据、地址, 指令等信息的路径称为数据通路, 要了解计算机是怎么工作的, 必须了解计算机中的数据通路。

#### (4) 组合逻辑控制器例题分析

**【例 1-57】** 计算机控制器常用有几种方案？说明组合逻辑控制器的特点。

解：控制器是计算机的控制中心，负责向各个组成部件提供操作控制命令，因此是非常复杂的，特别是操作控制部件负责提供全部指令执行时所需的全部控制命令。

实现操作控制部件通常使用两种方案：组合逻辑控制方案和微程序控制方案。

组合逻辑控制方案，是根据每个控制命令产生的条件，如什么指令，什么时间，什么状态，把所有产生条件“与”起来表示这个控制命令，再将同类的控制命令收集到一块，写出其逻辑函数表达式，用与门、或门、非门基本逻辑电路实现。

这种方案非常复杂，很不规整，难以检查调试，出错时难以修改，扩充指令时也需要从修改逻辑表达式开始，到重新生产印刷电路板。这种方案的好处是形成控制命令经过门的级数比较少，执行时比较快，在要求速度的计算机中经常采用。

#### (5) 微程序控制器例题分析

**【例 1-58】** 说明微程序控制器原理和特点。

解：微程序控制器与组合逻辑控制器的最大不同处是产生操作控制命令的方法不同。

一条指令的执行过程可分解为若干周期，若干节拍来实现，一条指令的操作可分解为若干个基本的操作序列，每一个基本操作序列可用一条微指令来表示，因此一条指令可用若干条微指令组成的微程序来实现，微程序放在被称为控制存储器的专门存储部件中。

执行一条机器指令时，可通过逐条执行从控制存储器中取出的微指令来实现。

微程序控制器的特点是：逻辑结构整齐，修改指令、扩充指令方便，只要修改或增加对应机器指令的微程序就行，在大型机器中广为采用，缺点是执行一条机器指令要通过执行多条微指令实现，速度较慢。

**【例 1-59】** 什么是微程序？机器指令和微程序有什么关系？

解：微程序是实现机器指令功能的微指令序列。由于每一种机器指令的功能都是不同的，因此其对应的微程序也是不同的。

每一种机器指令都有其确定的微程序与其相对应，可由机器指令的操作码作为进入对应微程序的入口。修改扩充机器指令时，只要修改增加其对应的微程序即可。

**【例 1-60】** 什么是微命令？什么是微操作？在机器中起什么作用？

解：在计算机中控制命令的最小单位称为微命令，如水平型微指令中每一个二进制位表示的控制信号，又叫操作控制命令。

一个微命令对应完成的操作叫做微操作，是计算机各个部件的最基本操作，如控制数据传送的开门电位，寄存器接收数据的打入脉冲等。

**【例 1-61】** 控制存储器有什么作用？计算机对控存的要求是什么？

解：存放机器指令微程序的存储器叫控制存储器，简称控存。控存单元的字长应该



存放一条微指令，机器执行微程序时，每次取出一条微指令，每拍执行一条微指令。

控存应该用 ROM 半导体存储器实现，因为机器指令的功能在设计机器时就已经规定了，并且不能随变改动，因此，微程序的内容在设计完成后就不能再改动了，为了保证机器指令工作可靠性，保证微程序的可靠性，用只读不写存储器来实现是一种最好的选择。但在实验性的机器中有时也可采用 EPROM，为设计者调试微程序提供修改的方便。

当然 CPU 对控制存储器的最高要求是速度要快，因为完成一条指令要通过完成多条微指令来实现的，如果控存存取周期很长，取微指令就要占用很长时间，当然完成一条指令的时间就更长了，因此控存速度是限制广泛采用微程序方案最重要的因素。现在控存都采用高速的半导体存储器；但对于高速计算机来说仍感到控存速度是一个瓶颈。

水平型微指令，操作并行度高，微指令字的字长较长，速度较快，但对固存来说其位码利用率较低。

### 3. 存储系统例题分析

存储器是存放被运算数据和程序的关键设备，存储器的速度和容量一直是人们努力解决的核心问题。除了提高主存单元的速度外，人们提出了高速缓存、主存和外存储器的三级结构，在现实情况下，给出了一种较好的解决方案。

#### (1) 存储器的特性例题分析

主存是 CPU 可以直接随机访问的存储空间，使用者对主存储的要求是速度、容量、带宽以及可靠性。

为了解决主存成本高、容量小的问题，人们提出了各种外存储器方案。如磁盘、光盘等设备具有大容量、低成本的特点，但速度太低，CPU 不能直接访问它们，不能从外存一个一个地取出数据或一条指令，只能作为主存的辅助存储器。

为了提高主存的速度，人们又提出了高速缓冲存储器方案。高速缓存、主存、外存紧密配合构成一个完整三级存储系统，较好地满足了人们提出的要求。

#### 【例 1-62】说明主存储器的特点和速度、存储容量的含义。

分析：主存储器是随机访问存储器，CPU 可以直接访问存储器任何一个单元，且其访问速度与访问单元的位置无关。这种随机访问存储器简称 RAM。

机器对主存的要求首先是速度要快，通常用存取周期来表示。一个存取周期，包括读写时间和恢复再生时间。因此存取周期是指连续两次访问同一个存储单元的最小间隔时间。

存储容量是 CPU 能够直接存取的存储器单元数的总和。CPU 每访问一次主存，读出的单位是一个字，字的长度不同机器有不同的规定，但都是字节的整数倍。在一些机器中，访问一次主存，可以读出一个字节的数据，因此每一个字节必须有一个地址，称为字节寻址，其存储器的容量单位是字节 (byte)。

主存的容量决定于由指令寻址方式产生的操作数有效地址的位数。若有效地址是 10 位二进制数据，则其容量为  $2^{10} = 1K$ ；若有效地址是 20 位数，则其容量是  $2^{20} = 1M$ ；若

有效地址是 30 位二进制数，则其容量是  $2^{30} = 1\text{G}$ 。

**【例 1-63】** 某机字长 16 位，存储器存取周期是 500ns，问存储器的带宽是多少？

解：带宽的含义是指每秒种由存储器读出的二进制数据的位数。因为存取周期是 500ns，一秒钟可以访问 RAM 的次数为  $\frac{1}{500\text{ns}}$ ，而每访问一次 RAM，读出 16bit 数据。

$$\begin{aligned}\text{它的带宽} &= 16\text{bit} \times \frac{1}{500\text{ns}} \\ &= 16 \times \frac{1}{100 \times 10^{-9}} \\ &= 16 \times \frac{10^9}{100} \\ &= 16 \times 10^7 \\ &= 1.6 \times 10^8 \text{ bit/s} \\ &= 32\text{Mbit/s}\end{aligned}$$

**【例 1-64】** 说明下面各种型号半导体存储器的名称和特点。

RAM、DRAM、SRAM

ROM、PROM、EPROM、E<sup>2</sup>PROM、FM

解：RAM 为随机读写存储器。

DRAM 动态随机存储器，采用动态 MOS 电路，需要定时刷新保存当前信息。单管 MOS 电路集成度高、功耗小，是当前 RAM 的主流型号产品。

SRAM 静态随机存储器，不需刷新，速度快，但线路复杂集成度低，写入数据后只要不断电，即可长久保持数据不变。

ROM 只读存储器，用于存放常用的固定程序，只读不写。

掩模 ROM 出厂时厂家已经把 ROM 制好，用户不能改动。

PROM 可编程只读存储器，用户只能编程一次写入所需数据，以后不能再改动。

EPROM 可擦除可编程 ROM，用紫外线照射 15 min 擦除，再用专门装置写入新内容。

E<sup>2</sup>PROM 为电可擦除可编程 ROM，使用比 EPROM 方便。

FM (FlashM) 又叫闪存。为快速联机可改写只读存储器，但擦除时只能成块擦去。

**【例 1-65】** 说明闪速存储器 FlashMemory 主要特性与 E<sup>2</sup>PROM 有何差别？

闪存又叫快擦存储器。属于一种新型电可擦除可编程、非易失性存储器，当机器断电后，闪存中的信息像磁盘一样长期保存，而速度又快于磁盘，体积小、可靠性高，擦除时速度很快、但只能按数据块整块擦去，是优良的磁盘代用设备，又称硅盘。

闪存与 E<sup>2</sup>PROM 相像，都是属于电可擦除的可编程的只读存储器，闪存虽只能按数据块整块擦除，但擦除时间比 E<sup>2</sup>PROM 快，读出时间也快，小于 90 ns 可代替 ROM 使用。

E<sup>2</sup>PROM 的擦写过程分为两步进行，先擦除该单元原有数据，再在下一个写周期中将新的数据写入，写操作允许信号在 10ms 以上，速度较慢。



## (2) 主存储器的组成和工作原理例题分析

**【例 1-66】** 用 16K×4 位的 RAM 芯片构成 64K×4 位存储器需多少 RAM 芯片？需多少根地址线？多少根数据线？片选线有何用途？地址线、数据线如何连接？

解：需要 RAM 芯片数为  $\frac{64K \times 4 \text{位}}{16K \times 4 \text{位}} = 4 \text{片}$

芯片连接方法属于字扩展方式。

存储器容量 64K =  $2^{16}$  共需 16 根地址线。存储器字长 4 位，共有 4 根数据线。RAM 芯片 16K，具有 14 根地址线；但 RAM 芯片中有一个片选信号  $\overline{C_s}$ ，主要用来扩充存储器容量。

当  $\overline{C_s}$  为低电位时，该片才能工作。

当  $\overline{C_s}$  为高电位时，禁止该片工作。

扩充容量时，存储器的地址线位数比 RAM 芯片上地址位数多，可将多出的高位地址线经过译码器输出分别控制不同 RAM 芯片的片选端  $\overline{C_s}$ ，以达到扩充容量的目的。

如本例中存储器容量为 64K×4 位，而每片存储容量为 16K×4，64K×4 的存储器必须用 4 片 16K×4 的 RAM 片子来组成；每片负责存储总容量的 1/4，各片按照高位地址要求轮流工作，每一时刻只有一个片子工作。因此根据多出的高 2 位地址用译码器将高位地址译出 4 个信号 00、01、10、11 分别控制 4 个片子的片选端工作。

地址连线方法：片内地址线按对应地址并联，高 2 位地址接 2-4 译码器输入端，译码器输出端分别与不同存储芯片片选端  $\overline{C_s}$  相连，决定 4 个芯片中读写哪一个芯片。

数据线连法：由于 4 个芯片都是平等的，都可以读出存储器 4 位数据，因此 4 个 RAM 芯片的数据端按对应位并联。

**【例 1-67】** 什么叫存储器刷新？什么类型存储器需要刷新？为什么？

解：存储单元电路的趋势是采用 MOS 电路，由于集成度高，功耗小受到用户欢迎。

但集成度最高的是单管 MOS 存储器，由于每一位二进制存储单元只用一个 MOS 晶体管和一个存储电容来实现，依靠电容上是否存储电荷来记录存储“0”、“1”信息。但电容上的电荷时间久了，例如超过 2 ms 会逐渐泄漏，丢失存储的信息，为了保证存储器工作可靠性必须在存储电荷泄漏前对每个存储单元进行补充充电，这种情况称为刷新，一般规定对动态存储器 DRAM 必须在 2 ms 之内对所有单元刷新一遍。

**【例 1-68】** 说明主存储器的组成和读写工作原理。

解：主存储器是按地址访问的随机访问存储器，存储矩阵中所有存储单元的地位都是平等的，因此要求访存时对每一个存储单元都同时进行地址译码。

主存是按单元的地址进行读写的，因此，存储器中应该有地址寄存器。存放数据单元的集合称为存储体。由地址寻找存储体中某一个存储单元时是通过地址译码电路实现的。读出的数据存放在数据缓冲寄存器中，准备送往数据总线，所有这些操作都是由存储器读写控制电路控制实现的。



写入存储器的操作与读出不同,除了 CPU 提供存储单元的地址外,还必须在写入前将要写入的数据通过数据总线送往存储器的数据缓冲寄存器,再在写入命令控制下将数据写入到地址寄存器指定的存储单元中。

因此主存储器必须包括:存储体、地址寄存器、地址译码器、数据缓冲寄存器和读写控制电路。

**【例 1-69】** 说明存储系统中三级存储系统的作用原理及实现方法上的异同。

解:三级存储系统的主要目的是在现有技术基础上构造一个高速度大容量的存储器,其速度如同访问寄存器,其容量和磁盘系统一样大。

其原理是基于程序访问的局部性。当执行程序时,其后继指令和所需数据都是相对集中存放的,因此只要把包括当前指令的一个程序段一起取到高速存储器,则访问后继指令或数据时,很大可能性是这些指令和数据已经取到高速存储器中,CPU 访问时可以很快取走,以提高运算速度。

主存是一个 CPU 随机访问的存储器,但其速度比 CPU 慢一个数量级。为了使主存与 CPU 速度匹配,特别在 CPU 和主存之间构造一高速缓冲存储器 cache,把正在执行的程序段调入高速 cache 中,其后 CPU 再取指令和数据,大部分可在 cache 中找到,就不用再访问主存了,因而提高了 CPU 的访存速度。

用户程序可能是很大的,主存容量又满足不了要求,为了扩大存储器的容量,特别又增加了一个大容量的辅助存储器,用来存放暂不使用的程序和数据。当 CPU 需要使用辅存中的程序时,再将有关程序成批调入主存储器中,由于程序的局部性原理,CPU 需要取后继指令或数据时,只要访问主存就可以得到了,不需要访问辅存,因而大大提高了访问辅存的速度,大大扩充了主存的容量。

理想情况下,如果 CPU 每次访问辅存中的指令或数据都可在主存中得到,每次访问主存单元都可在高速缓冲存储器 cache 中找到,则我们好像得到了一个速度与 cache 一样快,容量与辅存一样大的主存储器,这是非常理想的。

cache—主存层次和主存—辅存层次的工作原理都是相似的。在访问时,都需要经过地址变换,每次交换数据都是以数据块为单位,都是利用程序局部性原理,如果每次访存 cache 命中率较高,则我们将得到一个速度比主存快,容量比主存大的存储器,这是我们期望的。

因为 cache—主存层次更强调提高速度,所以地址变换等机制都是采用硬件办法实现的。而主存—辅存层次更侧重于扩大容量,并不苛求于速度,所以地址变换等机制采用软件方法实现。

### (3) 高速缓冲存储器例题分析

**【例 1-70】** 设有一个存储器,容量是 256KB,cache 容量是 2KB,每次交换的数据块是 16B。

求:① 主存可划分为多少块?

② 主存地址多少位？cache 地址多少位？

③ cache 容量 2KB，可划分为多少块？

④ CPU 访问 cache 进行地址映像时，主存地址分为几个部分？每部分多少位？

解：① 主存容量 256KB，每块 16B，主存可划分的块数为： $256\text{KB} \div 16\text{B} = 16\text{K}$  块。

② cache 容量 2KB，每块 16B，cache 或划分的块数为： $2\text{KB} \div 16\text{B} = 2048 \div 16 = 128$  块。

③ 主存容量 256KB， $2^{18} = 256\text{K}$ ，主存地址位数为 18 位，cache 容量为 2KB， $2^{11} = 2\text{K}$ ，cache 地址有 11 位。

④ 访问 cache 时，主存地址可划分为 3 部分：区号、块号、与块内地址。

区号即主存地址高位标志，其位数是主存地址位数与 cache 地址位数之差。

本例中高位地址标志（区号）=  $18 - 11 = 7$  位。

也就是说，整个主存可以分为  $2^7 = 128$  个区，每个区的容量相当于 cache 的容量，（即 2KB）。

块号：cache 中可分为 128 块， $2^7 = 128$ ，块地址编号 7 位。

块内地址，决定每块大小， $16 = 2^4$ ，每块 16 字节，块内地址 4 位。

【例 1-71】什么叫 cache 命中率？和哪些因素有关？如果 CPU 执行一段程序，访问 cache 3800 次（即  $N_c$ ），访问主存 200 次（即  $N_m$ ），cache 的存取周期  $T_c = 50\text{ns}$ ，主存存取周期  $T_m = 250\text{ns}$ 。

求 cache 命中率，平均访存时间及 cache—主存系统效率。

分析：CPU 正在执行的一段程序已取入 cache 中，当 CPU 访问某个存储单元的指令时，该单元已保存在 cache 中，CPU 可直接由 cache 中读出该条指令，我们称之为 cache 命中。

cache 的命中率是指一段时间 CPU 访问 cache 的次数与 CPU 访问主存与 cache 次数之和的比值。

cache 命中率越高，CPU 访问存储器取数的速度越快、运算速度也越快。

cache 的命中率与 cache 容量大小有关，与每次交换数据块大小有关，还与 cache 的替换算法、地址变换方法有关。

若一段时间中，CPU 访问 cache 的次数  $N_c = 3800$  次，访问主存的次数  $N_m = 200$  次，则 cache 命中率为：

$$H_c = \frac{N_c}{N_c + N_m} = \frac{3800}{3800 + 200} = \frac{3800}{4000} = 0.95$$

平均访存时间：

$$\begin{aligned} T_a &= T_c H_c + (1 - H_c) T_m \\ &= 50\text{ns} \times 0.95 + (1 - 0.95) \times 250\text{ns} \\ &= 50\text{ns} \times 0.95 + 250\text{ns} \times 0.05 \\ &= 60\text{ns} \end{aligned}$$

cache—主存系统效率为 cache 的存取周期和 cache—主存系统平均访存时间之比：

$$e = \frac{T_c}{T_a} = \frac{500\text{ns}}{60\text{ns}} = 83.3\%$$

**【例 1-72】** cache 地址映像有哪几种方式？各有什么特点？

**解答：** CPU 访问主存时，给出主存单元的地址，在具有 cache 高速缓存的计算机中，首先必须检查该主存单元的内容在 cache 单元中有没有，如果已经调入 cache，我们可直接由 cache 中取出，不需要再访问主存。如何把主存单元的地址变成 cache 相应单元的地址，叫地址映像。

cache 地址映像通常有三种方式：

### ① 直接映像

因为主存容量比 cache 大，可把主存分成若干与 cache 容量那么大的区，如果 cache 容量是 2K 单元（地址 11 位），主存容量 1M 单元（地址 20 位），则可把主存分成  $2^{20-11} = 2^9 = 512$  个区。cache 中又分成若干块，主存中也分成相同大小的块，显然主存每区中的块数与 cache 中的块数是相等的。每个数据块都有块号地址。

访问主存时，向 cache 中传送数据时，一次读出一个数据块。

直接映像方式规定、主存中某区中某个块读出时，其内容需调入与 cache 中块号相同的那个块中，为了区别是哪个区的数据块放在 cache 中，必须在该块数据调入 cache 的同时，将该数据块在主存中区号（高位主存地址、或叫主存字块标志）也存入 cache 对应的数据块标志中。

直接映像方式、地址转换方法简单，即主存中各区的 0 块只能放在 cache 中 0 块，主存中各区的 1 块只能放在 cache 中的 1 块……。主存各区数据块调入 cache 时，其块号必须与 cache 的块号相同，不能任意存放。其缺点是 cache 单元利用率不高。因为块号相同的多个主存数据块，只能放在 cache 同一个数据块中，其他 cache 数据块空闲也不能用。

### ② 全相联映像

它允许主存中任何数据块存放到 cache 空间中任何一个数据块位置上。存放起来很灵活，缺点是访问主存时，查找 cache 非常困难，它必须把 cache 中每一个数据块的高位地址标志读出来与要访问的主存单元高位地址相比较，这是很复杂的，也是很慢的，为了提高查找速度，通常采用按内容访问的联想存储器，当然这种设备是很复杂的。

### ③ 组相联映像

是直接映像与全相联映像的折中方案，其映射方案是：把主存与 cache 都分成若干组，每组又分成若干块。主存分组与 cache 分组间采用直接映像方式，主存与 cache 交换时以数据块为单位。组内各字块采用全相联映像，即在指定分组内各数据块是随意存放的。

组相联映像在学习性与复杂性上都界于直接映像与全相联映像之间。

#### (4) 虚拟存储器例题分析

**【例 1-73】** 什么叫虚拟存储器？什么叫虚拟地址？什么叫虚拟存储空间？为什么要构造虚拟存储器？

**解：**用户希望主存的空间大一些，以便存放更大的应用程序，实际上做不到这一点。磁盘容量很大，但速度很慢，不适合 CPU 直接由磁盘中存取指令和数据。根据程序访问的局部性原理，可把磁盘作为辅助存储器使用，存放当前 CPU 暂不使用的程序和数据，等到 CPU 需要使用有关程序时，再把它们成批地由磁盘调入主存，CPU 访问主存取出有关指令和数据比访问磁盘快得多。因此在理想情况下用户得到一个容量如同磁盘一样大，速度如同主存速度一样快的实际上并不存在的新的存储器，称为虚拟存储器。包括磁盘在内的整个存储空间称为虚拟存储空间，访问虚拟存储空间的地址称为虚拟地址或逻辑地址，而原来的主存储器称为物理存储器（实存），其地址称物理地址，其容量称物理存储空间。

实现虚拟存储器的关键是要建立主存与磁盘间程序块自动装入和替换的机制，自动实现虚拟地址与物理地址的转换。这种机制是由软件完成的。

**【例 1-74】** 什么叫页式虚拟存储器？什么叫页表？说明工作原理。

页式虚拟存储器把虚拟存储空间（逻辑空间）和主存空间（物理空间）等分成固定大小的页面，虚存和实存间交换数据是以页面为单位进行的，每个虚拟页面可以装入主存中任一个物理页面。如果页面大小为 4KB，则页内地址共 12 位，虚拟空间的虚拟地址，去掉页内地址（如上例为 12 位），就是页面地址简称页号，即虚拟地址（逻辑地址）的高位地址。

给定一个虚拟地址（逻辑地址）访问虚拟存储器取出该单元的数据的过程，首先必须查页表，进行虚实地址转换。页表中存放各逻辑页面调入主存时的对应物理页号。如果所读逻辑页面已调入主存，可按页表给出该页面装入主存的物理页号作为访问主存的高位地址，逻辑地址的页内地址作为物理页面的页内地址（被访问主存单元的低位地址），二者合起来构成主存的物理地址。

如果虚存访问的页面还未调入主存，则必须先将磁盘中该页调入主存一个空闲页面中，并在页表中填入有关物理页号。

**【例 1-75】** 什么叫段式虚拟存储器？什么是段表？说明其工作原理。

**解：**按照用户程序的逻辑功能，将程序划分成若干个独立的程序模块，这些具有独立功能的程序部分称为程序段，简称段。段作为独立的功能单位可以被其他程序调用。因此在主存与辅存之间调度数据时，以段为单位传送是合理的。

依照页式虚拟存储器，先将辅存中的用户程序分成若干段，仿照页表，设计一个段表，段表给出每段程序调入主存时的存放位置，包括该段的起始地址、段长、以及装入特征位等信息。这种虚拟存储器中，程序调入主存时也按段划分，这种存储管理方式称段式管理。



段式管理方式中, 每个程序段都有独立逻辑功能, 便于多道程序共享, 但各段长度不同, 各段的存储位置如起点、终点不固定, 给调度时分配主存空间造成很大困难, 也容易在主存中留下无用的碎片, 造成浪费。

段式虚拟存储器的工作原理与页式虚拟存储器管理类似, 不同的是传送数据单位是段, CPU 访问虚存时提供虚拟地址, 先进行虚实地址转换, 需要先查段表, 找出该段在主存中的起始位置, 再加上虚拟地址中低位部分提供的段内地址, 其和即为主存的实际地址 (物理地址), 按实存地址访问主存, 即可达到 CPU 的访存要求。

当然 CPU 访问的段不在主存中, 还需要将辅存中有关程序段调入主存中一个空闲的连续存储区中, 并填写段表, 说明该段在主存中的起始地址和段长, 装入位置“1”。CPU 要读出该段内容时, 需把该段在主存中的起始地址加上虚地址低位提供的段内地址, 即可得到要访问的主存单元地址。

#### 4. 外围设备

计算机外围设备包括输入设备, 输出设备和外存储器。有时也把数字通信设备包括在内。

##### (1) 外围设备的种类和特性例题分析

**【例 1-76】** 说明输入设备和输出设备的作用。

**解:** 输入设备和输出设备是计算机与外部世界交往的桥梁, 简称 I/O 设备。

外部世界千变万化, 表现形态、工作速度迥异, 如何与计算机交往, 都是需要 I/O 设备解决的难题。

输入设备需要把外部信息, 包括数字、文字、声音、图像转换成二进制编码输入到计算机存储器中保存, 等待运算器处理。

输出设备需要把计算机内的二进制编码转换成人们容易辨识的数字、文字、声音、图像等信息传送给人们。

**【例 1-77】** 说明外围设备工作的特性。

**解:** ① 外围设备种类繁多, 低速、中速、高速设备花样翻新, 但共同的特点是设备本身工作相对 CPU 的工作是独立的。设备与 CPU 通信是异步的, 通常采用应答方式传送。用请求允许信号开始传送, 用回答信号表示传送结束, 中间传送时间长短不是固定的。

② 有些外围设备作为控制设备使用时, CPU 必须经常检测设备状态, 及时采取果断措施, 避免事故发生。有些高速设备传输的数据, 要求 CPU 及时处理, 以免丢失, 因此要求实时处理。

③ 外围设备类型特性不同, 设备工作原理不同, 产生信息的方法不同, 数据格式不同, 因此与 CPU 通信时, 接口的结构、功能不同, 为了简化计算机的接口设计, CPU 采用标准接口的办法与 I/O 交往, 要求 I/O 设备也要遵照标准接口的规定与主机通信。

##### (2) 显示设备例题分析



**【例 1-78】** 什么是图形？什么是图像？图形显示器与图像显示器二者有什么差别？

解：图形是指用点、线、面、体生成的几何图形，如电路图、机械零件图、建筑工程图等，图形中没有亮暗层次差别，是一些线条图。

图像是指用录相机拍下来的照片、录像等具有亮暗层次的照片，计算机中要表示图像首先要将照片幅面上连续的亮暗变化转换为点阵位图中每个像素的数字量，以点阵的方式存入计算机中或显示出来。

表示图形只要在有线条地方用坐标或公式表示出来，其他地方作为背景不需要指明。占用的存储空间小，对显示器的要求不高。

表示图像需要把平面上每个像素（光点）一个不漏地表示出来，指明每个像素颜色的深浅层次，因此占用的存储空间较大，对图像显示器的像素数目和扫描速度都有一定的要求。

**【例 1-79】** 说明字符显示器，图形显示器，图像显示器的异同和特点。

解：三种显示器都是采用类似电视的原理，采用电子束逐行扫描，显示出不同的对象。

字符显示器是专门用来显示 ASCII 码表示的西文字符，显示时必须由 ASCII 码找出其对应的字符形状，一般用字符发生器给出字符形状的点阵信息，通过电子束逐行扫描，显示出有关字符。

图形显示器可以显示各种点、线、面、体组成的几何图形，如电路图、机械零件图、建筑图等，屏上显示的都是线条图，图形没有亮暗层次，当然显示不同的图形也是利用点阵位图实现的，不同的是表示每个光点（像素）的方法比较简单。

图像显示器，可以显示有亮暗层次变化的照片录像，显示器原理要复杂些，表示不同图像也是利用点阵位图实现的，与图形显示器不同的是表示每个光点（像素）的亮暗层次要用多位二进制数表示，因此还要用灰度级来决定每个像素的亮暗级别、亮暗程度或颜色种类。

**【例 1-80】** 说明显示器中分辨率的意义，表示显示器分辨率的指标是什么？提高分辨率技术上要采取什么措施？

解：显示器的分辨率指荧光屏所能显示的光点数目，即像素的多少。显示器的像素越多，显示的图像越清晰。

表示分辨率的指标，通常用行数×每行像素数目表示，如分辨率为 640×480，表示每屏 480 线，每线 640 个像素。满屏共有 640×480 像素。也有用两个像素中心间距表示，0.31mm，即表示两个像素中心间之间隔为 0.31mm，像素中心间距越小，表示相同长度内像素数目越多，图像应该越清晰。

提高分辨率，要求荧光粉细，电子束要细，偏转控制要精确，显存容量要大，速度要快。

**【例 1-81】** 显示器中灰度级的概念是什么？用什么方法表示灰度级？

解：灰度级用来表示图像显示器中每个光点的亮暗级别或颜色种类，灰度级别越多，表示图像级别的层次越细腻，越逼真，当然表示每个像素的灰度级的二进制数字位数越多。

例如：一个像素的灰度层次分成 16 级，则必须用 4 位二进制数表示它，该像素的灰度可以分别是 0 级、1 级、2 级……最高 15 级，共 16 个级别。

如果每个像素的灰度级用 8 位二进制数表示，则它可表示  $2^8 = 256$  种灰度级，因此表示该像素的灰度时，可用 256 级的任何一级来描述。

当然灰度级越多，表述一个光点的亮暗层次的二进制数的位数越多，占用的存储器容量越大。

黑白显示器中每个像素的灰度分为二级，用 1 位二进制数表示该光点是亮或不亮。

**【例 1-82】** 为什么要对荧光屏进行刷新？刷新频率是多少？

什么是刷新存储器？刷新存储器的容量与什么有关？如果一个显示器的分辨率为  $1024 \times 1024$ ，灰度级为 256，求刷新存储器的容量。

解：图像显示器利用电视中 CRT 电子束扫描原理制成，电子束扫过后光点亮度只能保持几十毫秒，为了保持眼睛看到稳定的不闪烁的图像，屏上每个光点必须在其亮度消失前第二次再重复显示一次，这种办法称为刷新。电视制式中规定整屏图像刷新频率是每秒 50 帧，为了提供整屏图像的信息，必须将被刷新的内容保存在一个专门存储器中，这一个存储器称为刷新存储器或视频存储器 VRAM。刷新存储器的容量与图像分辨率和灰度级有关。

若显示器的分辨率为  $1024 \times 1024 = 1\text{M}$  像素，每个像素有 256 级灰度，则每个像素须用 8 位二进制数来表示。因此其刷新存储器的容量为  $1024 \times 1024 \times 8\text{bit} = 1\text{MB}$ 。

**【例 1-83】** 说明字符显示器与字符发生器的关系。

解：字符显示器可用来把计算机中存储的字符编码——ASCII 码，经过转换变成字符的书写格式显示出来。

字符显示器采用图形的方式，把字符表示成字符点阵，由电子束逐点逐行扫描显示出来。其关键部件是字符发生器。

字符发生器中存储有英文大小写字母、阿拉伯数字等可显示符号的字形点阵编码，如果每个字符图形可用  $5 \times 7$  的点阵来表示，则每个字符点阵用 8 个字节的编码即可表示这个  $5 \times 7$  的字符点阵信息。每一个英文字母可用 1 字节的 ASCII 编码来表示，以便计算机中处理、存储、传送；也可用其 8 字节的点阵图形信息来表示，以便显示打印时，把 ASCII 编码转换成字符对应的书面形状显示出来，方便使用者阅读。

因此字符发生器就是个转换装置，当需要显示字符时，首先根据该字符的 ASCII 码作为地址查询字符发生器，根据字符发生器给出的图形点阵信息，由电子束逐点逐行地进行扫描显示出字符对应的字形。

(3) 外存储器例题分析

**【例 1-84】** 说明磁表面存储器磁头读写原理。

解：磁表面存储器是利用铁磁材料的剩磁特性来保存信息，磁性材料有两个不同状态，分别用来表示“0”和“1”两个不同的信息。

磁性材料涂覆在介质表面，由磁头对载磁体进行读写操作。

磁头是一个绕有两个线圈的电磁铁。写入时，当写线圈中通入正向电流，磁头铁心中产生一定方向的磁通，通过磁头下方间隙处产生的磁场将载磁体的一个小区磁化成某一方向，写入电流消失后，该小区仍保持剩磁状态，表示写入“1”，当写入线圈通入反向电流，磁头下方间隙处产生的反向磁场，将载磁体的一个小区磁化成相反的方向，我们称其剩磁状态为“0”。

当读出时，让载磁体和磁头做相对运动，磁头读线圈切割载磁体小区的剩磁磁场，会在磁头读线圈中产生感应电流，经过放大整形，可分别读出保存在载磁体表面磁层中保存的“0”或“1”的不同信息。

**【例 1-85】** 说明调相制与调频制磁记录方式的特点。

解：调相制记录方式规定用写入电流相位不同变化，表示写入不同的代码。

例如，在写周期中间当写入电流由低电位跳变为高电位时，表示在载磁体中写入“1”；反之写入电流由高电位跳到低电位时，我们称在载磁体中写入“0”。

当连续写入多个“1”或多个“0”时，则在记录单元边界处规定写入电流必须改变方向，以保证连续写入的两个“1”都是由低电位跳到高电位所致。在读出信息时，必须在记录单元中心处读出的才是存储的信息。

调频制记录方式规定，利用每个信息周期内写入电流改变方向的频率不同，表示写“1”或写“0”。

目前通用倍频制，写入电流变化规律是：当记录信息周期开始时，写入电流方向改变1次，写“1”时在记录信息周期中间，写入电流方向再改变1次，但写“0”时信息周期中间写入电流方向不改变。因此在读出时，若信息周期中间读出信息称读“1”，信息周期中间无读出信号，称为读“0”。

调相制调频制都不是利用幅度鉴别读“1”或读“0”信号，而是利用相位或频率不同决定读出是“0”或是“1”，对信号噪音比要求低，工作比较可靠，另外在每个记录周期开始或中间都有信息读出可以作为读出同步脉冲，称为自同步，不需再设置同步脉冲。

**【例 1-86】** 说明磁盘存储器记录密度、存储容量的含义。说明扇区，格式化存储容量及非格式化容量的含义。

解：磁盘存储器记录密度指盘片表面单位面积上记录的二进制信息个数，通常用位密度和道密度表示。

位密度指沿着磁道方向，单位长度存储的二进制信息个数，单位通常用每英寸多少位，或每厘米多少位。

道密度指沿着磁盘半径方向，单位长度存储的磁道的数目，单位通常用每英寸多少磁道，或每厘米多少道。



在磁盘存储器中,以磁道为单位进行读写单位太大,存取不方便,一般要把磁道分成若干扇区,以扇区为单位进行读写。磁盘上每个磁道的扇区数目都是相同的,每个扇区存储的信息位数是相同的。由于磁盘内磁道的长度比外磁道短,因此内磁道存储信息的位密度较大,说某磁盘位密度时,一般指内磁道最大位密度。

计算磁盘每片存储容量时,如果给出一个磁盘表面上有几个磁道,每道分几个扇区,每个扇区存储多少位二进制信息,则该盘片的存储容量为每面磁道数 $\times$ 每道扇区数 $\times$ 每扇区存储的二进制信息个数。这种容量为格式化容量,是给用户使用时用户能够存储的信息容量。

如果知道磁盘的内外磁道半径、道密度、位密度,这时可计算该磁盘片上可存储的信息总量,包括有关地址、标志、校验信息,这时算出的是未格式化的所有存储信息的总量,称为非格式化容量。

**【例 1-87】** 已知一个盘组有 4 个盘片,其中 6 个是数据记录面,每面的内磁道直径为 22cm,外磁道直径为 33cm,最大位密度为 1600 位/cm,道密度为 80 道/cm。求该磁盘非格式化储存容量。

解:首先求出内磁道周长,算出每个磁道存储的二进制信息总量,再算出该面磁盘上磁道数目,可找到每个记录面存储的二进制信息数,如果求盘组非格式化信息存储容量,再用每片存储容量乘以记录面个数即可。

① 计算内磁道周长。

$$\pi D_{\min} = \pi \times 22\text{cm} = 69.12\text{cm}$$

② 一个磁道上能存储的二进制信息总量,最小的磁道周长 $\times$ 最大的位密度。

$$\begin{aligned} &= 69.12\text{cm} \times 1600 \text{ 位/cm} = 110592 \text{ 位} = (110592 \div 1024)\text{K 位} = 108\text{Kbit} \\ &= (108\text{Kbit} \div 8\text{bit})\text{B} = 13.5\text{KB} \end{aligned}$$

③ 计算每个记录面上磁道数目。

$$\begin{aligned} &\frac{\text{外磁道直径} - \text{内磁道直径}}{2} \times \text{道密度} \\ &= (33 - 22) \div 2 \times (80 \text{ 道/cm}) \\ &= 440 \text{ 道} \end{aligned}$$

④ 一个盘面能记录的二进制信息总量:每道存储二进制信息总量 $\times$ 磁道数。

$$\begin{aligned} &= 13.52\text{KB} \times 440 \\ &= 5940\text{KB} \end{aligned}$$

⑤ 盘组非格式化存储容量,每面存储信息数 $\times$ 记录面数。

$$\begin{aligned} &= 5940\text{KB} \times 6 \\ &= 35640\text{KB} \\ &= (35640\text{K} \div 1024\text{K})\text{MB} = 34.8\text{MB} \end{aligned}$$

**【例 1-88】** 什么叫扇区？什么叫圆柱面？磁盘和主机交换数据的单位是什么？什么是找道时间？什么是等待时间？什么是磁盘数据传输率？

**解：**因为磁道上存储的信息量太大，存取不便，一般把磁道分成若干个数据区，每个数据区存放相同容量的信息，每个数据区有自己的地址，磁盘和主机交换数据时，以数据区为单位，这样的数据区称为扇区，盘面上各个磁道上扇区数目相同，因此每个磁道上存储的信息量也相同。

因为固定头磁盘结构复杂、造价高，现在通常都采用活动头磁盘，即每个记录面只设置一个读写磁头，由读写臂带动磁头沿磁盘半径方向移动，寻找要读写的磁道，这段时间称为寻道时间。找到要求的磁道后，还需等待要读的扇区转到磁头下，才能开始读写操作，这段时间称为等待时间。

为了节省读盘时间，节省找道时间，希望把盘组中各记录面中相同的磁道组成一个存储区，我们称之为磁盘圆柱面，读写同一个圆柱面上各个磁道的数据时，不需寻找磁道，可以较快地读写磁盘数据。

磁盘数据传输率指的是每秒钟可读出的二进制信息数目。如果知道每个磁道上存储的数据字节数目，又知道磁盘的转速，即可求出磁盘的数据传输率。

**【例 1-89】** 已知磁盘分成 16 个扇区，每个扇区存储 512 个字节数据，磁盘转速 3600 r/min。求数据传输率。

**解：**① 一个磁道上存储的数据总数是  $512\text{B} \times 16 = 8192\text{B}$

② 求出磁盘 1 s 转几圈

$$(3600 \text{ r/min}) \div (60 \text{ s/m}) = 60 \text{ r/s}$$

③ 数据传输率

$$\text{每道数据数} \times \text{磁盘每秒旋转圈数} = 8\text{KB} \times 60 = 480\text{KB/s}$$

**【例 1-90】** 说明磁盘地址格式，寻址过程。

**解：**磁盘存储器的地址格式。

① 多个盘组时首先区别盘组编号。

② 一个盘组内部寻址时先找圆柱面号即磁道编号，当一个圆柱面上各记录面上的数据都存满时，再记到下个相邻磁道上。

③ 记录面号，即磁头编号，每个记录面有一个磁头。

④ 扇区号，扇区是磁盘与主机交换数据的基本单位。

因此访问磁盘某个扇区时，首先决定是哪个盘组工作。第二寻找工作磁道，要移动磁头到指定磁道位置，根据记录面号选择某个记录面上的对应磁头工作。第三寻找扇区，等待指定扇区转到磁头下方，才开始读写。

**【例 1-91】** 说明硬盘、软盘存储器的区别和特点。

磁盘存储器是容量大、成本低、速度快的非易失性大容量联机存储器。

硬盘的特点是盘片为硬质材料制成，磁头读写磁盘时不与盘片接触，盘片转速较高，



一般每分种几千转，因此读写速度快，数据传输率高。

软盘的盘片是用聚酯薄膜制成，盘片较软、磁头读写时采取接触方式，可缩小磁头与盘片间距，提高存储密度，盘片转速较低，容量较小，数据传输率低。软盘价格便宜，携带方便，微机中广为采用。

**【例 1-92】** 说明磁盘阵列 RAID0 模式的工作原理。

解：磁盘阵列 RAID 工作原理类似多体交叉存储器的工作原理，多个磁盘并行读写，提高磁盘存储器的存取速度和容错能力。

RAID0 工作模式类似多个存储体地址低位交叉编址，连续读出的多个数据，依次分别放在不同的磁盘中，几个磁盘交叉并行读取，不仅扩大了存储容量，主要是提高了磁盘数据存取速度。但 RAID0 没有容错能力，不像 RAID4、RAID5 增加了校验盘，把有关校验值写入校验盘，以便恢复故障盘中数据。

#### (4) 通信设备例题分析

**【例 1-93】** 为什么计算机通信中要使用调制解调器？

解：在计算机远程通信中，常常使用电话线作为通信介质，但电话线传送数字信号效率很低，且易引起失真造成通信错误。

为克服这种情况，常在被传送的数字信号上叠加一个正弦波，经电话线传送到目的地后再将叠加的正弦波分离出去，恢复为原来的数字信号，在传送过程中减少数字波形的失真，这个过程前者称调制，后者称解调制，完成有关功能的设备叫调制解调器。

调制时，用一定频率的正弦波作载波，用被传送的数字信号控制载波参数，通常可分别控制载波的幅度、相位、频率、形成三种调制方式。

### 5. 输入输出系统与控制例题分析

#### (1) 总线结构例题分析

**【例 1-94】** 什么是系统总线？为什么要用总线传送信息？说明总线的分类和特点。

解：总线是 CPU 与各个部件交换数据共享的通信线路。总线的特点是各个设备使用总线时必须是分时共享。

使用总线必须制定一套规则，规定总线的功能特性，连接方法，通信及控制方式等。

采用总线方式的主要优点是简化计算机的结构设计，减少设备间通信线路，便于系统的扩充和子系统的设计修改。

CPU 与存储器和外围设备通信的一组总线称为系统总线，系统总线包括数据总线、地址总线和控制总线三类。这种结构称为单总线结构，特点是结构简单，扩充容易。缺点是每一个时刻只允许一个设备占有总线，向总线上发送数据，往往成为计算机传送数据的瓶颈。

为了提高数据传输率，提高数据传输的并行性，在要求较高的机器中可采用双总线结构或多总线结构。

双总线结构把 CPU 与主存储器的通信总线称为存储总线或 CPU 总线。另外把 CPU

与 I/O 设备的通信总线称为 I/O 总线，两组总线可以独立工作。

**【例 1-95】** 什么是同步总线？有什么特点？

解：收发双方使用总线传送数据时，使用统一的时钟信号，按照约定的固定时间周期发送数据和接收数据，设备之间不需要应答信号。时序关系简单，实现也比较简单，但若设备速度偶然变化，将影响传输数据的可靠性。

同步总线适用于系统中各种设备操作的速度固定而且一致の場合，同步总线的长度不能太长。因为传送数据时不需要应答信号，同步传送的速度较快、效率较高。

**【例 1-96】** 什么是异步总线？有什么特点？

解：收发双方使用总线传送数据时，没有固定的收发时间，不能使用统一的固定的时钟来控制收发数据。

收发双方为了在总线上传送数据，必须提供专门发送信号和接收信号，才能可靠地完成数据传送工作。这种总线，称为异步总线。

异步总线的操作时间不是固定的，操作的每一个步骤都需要有专门的应答信号来表示。

在非互锁异步传送方式中，发送设备把数据放在总线上，然后向对方发“就绪”（ready）信号，告诉对方数据已在总线上。接收设备根据收到的 ready 信号，从总线上接收数据，并向发送方发“应答”（acknowledge）信号作为回答，通知发送方数据已经收到了。发送方收到接收方的“应答”信号后从总线上撤消数据，如此双方完成一次数据传送。

在全互锁异步传送方式中，使用四边沿协议，进行数据传送，工作更可靠，可适合各种速度 I/O 设备传送数据。发送设备把数据放到总线上后，向接收设备发“就绪”（ready）信号，接收设备接收数据后向发送方发“应答”信号，发送设备收到接收方的“应答”信号后复位“就绪”信号，“就绪”信号复位后接收设备才复位“应答”信号。

异步总线工作可靠，可以适应各种速度的设备传送数据，缺点是传送过程复杂，传送速度较慢。

**【例 1-97】** 微机总线标准中需要规定什么内容？为什么要采用标准总线？

解：总线是 CPU 与机器各部件进行信息交换的共享通路，包括与 I/O 设备通信。整机厂家和用户希望计算机能与各种厂家生产的 I/O 设备连接工作，各个厂家生产的 I/O 设备可以互连互换，I/O 设备生产厂家也希望自己生产的设备能与各种整机厂家生产的主机系统连接工作，因此希望整机厂家和 I/O 厂家都遵守统一的总线标准。为了适应这种要求，制定了总线标准，特别是微机总线标准，实用上显得更为重要。

微机总线标准包括总线的接插件标准，如每个引脚的功能名称、信号电平、负载能力、时序规范，以及插头座的尺寸、间距和定位要求。

采用标准总线优点是通用性好，便于开发、升级、更新系统，便于维修、备件购置等。

**【例 1-98】** 什么是 ISA 总线？什么是 EISA 总线？各有什么优点？

解：ISA 总线是 IBM 公司推出的与 IBM PC/AT 兼容的工业标准总线。可支持 CPU 为 Intel 80286, 80386, 80486 各种微机系统。总线时钟 8MHz, 最高传输率 16Mbps。

ISA 总线, 数据线 16 位, 地址线 24 位, 还有 12 根中断线, 14 根 DMA 与响应线等共 98 根引线。使用两个插槽。其中 62 线插槽与 IBM PC/XT 总线兼容, 新增加的 36 线插槽扩展了 XT 总线的性能, 如数据线、地址线的根数等。

EISA 总线是 COMPAQ, HP 等 9 家公司 1989 年提出的扩展的工业标准总线, EISA 总线是 32 位总数与 ISA 兼容, 包括 32 位数据线, 32 位地址线, 共有 196 个引脚, 总线时钟 8MHz。最高传输率 33Mbps

**【例 1-99】** 说明 RS-232C 串行接口四个应答信号的作用。

解：RS-232C 是一种串行总线, 通过调制解调器支持通过电话线进行远程通信。

RS-232C 用于实现 CPU 与一台设备串行传输数据, 该接口有 25 个引脚。其中：RXD 用来接收串行输入的数据。

TXD 用于向接收设备发送输出的数据。

RTS 用于向接收设备发出请求, 发送方请求接收设备接收数据, 这个信号简称请求发送, 是输出信号。是主方请求发送数据给对方。

CTS 称允许发送。接收方收到请求发送信号 RTS 后, 准备接收数据, 向发送方发出回答信号, 允许发送 CTS。对发送设备说是输入信号。

DTR 数据终端准备好, 表示接收设备(主方)请求对方发送数据, 是输出信号。

DSR 表示数据准备就绪, 表示发送设备准备好数据, 是发送设备对接收设备发出的 DTR 请求的回答信号, 表示发送设备可以发送, 是输入信号。

四个握手信号都是高电平有效(RS-232C 规定 +3V~+25V 为高电平逻辑“0”, 而 -3V~-25V 为逻辑“1”电平), 这种规定与 TTL 电路不同, 因此 RS-232C 与主机连接时还要进行电平转换。

**【例 1-100】** 说明外设总线 SCSI 的特点和用途。

解：SCSI 是小型计算机系统接口的英文名称缩写, 是一种系统级的输入输出总线。

SCSI-I 定义了总线时钟 5MHz, 8bit 宽度的总线, 最多可接 8 个设备, 各个设备可并行操作。是一种速度快、灵活性好的并行总线, 可用于磁盘、磁带、光盘等设备与主机通信。

主机通过 SCSI 适配器与 SCSI 总线相连。这种称为 SCSI 主设备的 SCSI 总线控制接口, 包括微处理器、数据缓冲存储器 RAM, 以及 DMA 控制器、协议控制器等。微处理器负责解释主机送来的命令和 SCSI 送来的信息, 负责主机与 SCSI 缓存, 缓存与 SCSI 总线间数据传送控制, 控制 DMA 操作。

SCSI 总线可以在同步方式下工作, 同步方式传输率为 5MBps, SCSI-3 型可达 320MBps。也可以异步工作。信号线 50 根, 8 条数据线和一位校验线, 可以单端方式



工作,其连线长度小于6米。也可作差分方式连接,提高抗干扰能力,连线长度可达25米。

## (2) 基本 I/O 接口组成和工作原理例题分析

**【例 1-101】** 什么叫接口? 接口有什么功能? 说明基本 I/O 接口的组成和分类。

解: 接口是外围设备与主机间通信的桥梁, 接口是设备的控制器, 或称适配器。

接口的主要功能是:

- ① 接收主机发来的命令, 控制外围设备操作。
- ② 反映设备工作状态, 以便主机发出不同的控制命令。
- ③ 作为 I/O 设备与主机间传送数据的缓冲, 暂时存放等待主机取走或设备取走的数据。

④ 中断逻辑, 当今多数设备以中断方式与主机通信, 接口中应设置中断控制逻辑。

接口基本组成包括: 设备选择电路, 数据缓冲寄存器, 控制与状态寄存器, 中断请求与屏蔽电路。基本 I/O 接口可分为并行接口和串行接口两种, 在串行接口中还应包括数据格式串并行转换电路。根据数据通信方式, 接口又可分为同步接口与异步接口。

**【例 1-102】** 说明异步串行通信的特点及传送一个字符时的通信格式。

解: 串行通信常用于远程通信和低速设备, 串行通信连线最少, 传送一个多位的数时, 每次发送一位数据, 分多次发送才能完成的。计算机内都是采用并行收发的, 因计算机要发送一个数据时, 常要通过移位寄存器, 把并行数据变成串行数据一位一位送出去。接收时也需把串行数据变成并行数据, 再送给计算机。

异步串行通信时, 因为接收器不知道何时传来的是有用数据, 因此在被传送的数据前要增加一个标志位, 当接收装置接收到这个标志时, 即可断定后面接收到的是有效的数据。一般数据字是8位二进制数, 事先必须约定, 数据位后面是1位结束位, 结束位固定为“1”电位, 有时结束位之前还可增加校验位, 这样构成一帧数据。

如果传送的两帧数据之间不是连续的, 则在前一帧数据字结束位之后, 下一帧数据字起始位之前插入若干个空闲位, 空闲位为“1”, 位数不定, 接收数据端不断地检测数据帧的起始位何时到来, 一旦发现数据线上的“1”电位变成“0”电位这个跳变到来, 即可认为起始位之后将是数据位, 准备接收数据, 接收端按事先约定的格式接收数据。

**【例 1-103】** 说明同步串行传送方式的特点。

解: 异步串行通信中, 每传送一帧数据(如一个字节)都需前头加起始位, 后头加结束位作为标志, 只有中间一部分是有效的, 两头附加的标志占用很多时间。

为了提高数据传送的效率, 可去掉这些标志, 发送端按规定的时间发送数据, 接收端按规定的时间接收数据, 采用统一的时钟控制收发, 去掉标志, 这种办法称为同步传送。

同步传送以数据块为单位, 数据块传送开始时还是需要用同步字符作为起始标志, 通知接收方准备接收数据, 字块传送结束时, 增加两字节的 CRC 校验码, 以提高传输



的可靠性。

同步通信速度快, 控制方法简单, 但发送时钟与接收时钟必须一致, 否则其误差可能带来不可靠的因素。

### (3) 输入输出程序查询方式例题分析

**【例 1-104】** CPU 与外设以程序查询方式传送数据的原理是什么?

解: 由于设备的工作速度很慢, CPU 不知道 I/O 何时把数据准备好送给主机, CPU 又怕 I/O 送来数据时又没有发现, 把数据丢了, 因此, CPU 不断地用测试指令查询设备的工作是否完成, 输入的数据是否准备就绪, CPU 执行循环程序一直监测设备接口中的状态触发器, 输入设备没有准备好输入数据时 CPU 就一直执行这种查询设备工作状态的程序, 直到输入数据准备好, 程序才转入输入指令, 取走要输入的数据。

当输出数据时情况也是类似的, CPU 一直执行查询程序, 测试输出设备输出的数据是否已经取走, 输出工作是否已经完成, 如果已经完成, CPU 才能再输出下一个数据。

显然在程序查询方式中, CPU 一直为 I/O 设备服务, 不能做其他事情, 因此 CPU 的工作效率是非常低的。

**【例 1-105】** I/O 设备与 CPU 通信时, 必须先对 I/O 设备进行寻址。现在常用的一种方法是将 I/O 接口与主存单元统一编址, 说明工作原理。

解: CPU 在输出工作时, 执行输出指令, 把 ALU 中寄存器的数据送到指定输出设备的接口的数据寄存器中。如果把该设备的数据寄存器作为主存的一个单元, 与主存统一编址, 则 CPU 可利用写主存的命令完成输出工作。

同理, CPU 输入数据时, 执行输入指令, 把指定输入设备接口中的数据寄存器中的数据取到 ALU 的寄存器中即可, 如果把该设备接口中数据寄存器作为主存一个单元, 与主存统一编址, 则 CPU 利用读主存指令即可完成输入工作。

这种方式的好处是可以省去 I/O 指令, 简化指令系统, 缺点是主存空间中有一部分单元指定给 I/O 接口寄存器, 缩小了主存的容量。

### (4) 中断系统例题分析

**【例 1-106】** 什么叫中断? 为什么要设置中断?

解: CPU 在执行程序过程中, 产生一些突发的偶然事件, 要求 CPU 暂停当前正在执行的程序, 转去为突发事件服务。CPU 为之服务完毕又自动返回继续执行原程序, 这个过程称为中断, 因为处理过程是用程序实现的, 又叫程序中断。

设置中断的原因。

① 提高 CPU 工作效率, 把 CPU 从查询 I/O 状态的等待过程中解放出来。办法是 CPU 启动 I/O 后, I/O 设备开始工作, CPU 自己继续处理原来的工作, 等待 I/O 工作完成后请求 CPU 取走数据。这种情况为 CPU 与 I/O 并行工作。

② 多台 I/O 并行工作, 提高 I/O 速度。

③ 解决实时处理问题, 生产过程中的异常情况, 计算机必须立即处理。

④ 计算机运行中出现故障, 必须及时处理, 因为已经出错了, 再继续工作下去也没意义, 等待排除故障后再继续运算。

⑤ 实现多机系统或网络环境下计算机间的通信要求。

⑥ 提供人机联系的手段。

**【例 1-107】** 主机在什么条件下响应中断?

解: 外部事件随机提出中断请求, CPU 也不是即刻响应的, 必须在一定条件下才能暂时停止现程序的执行, 转去处理中断请求要做的事。

① CPU 允许中断。CPU 在一些特殊情况下不允许中断, 大多数时候允许中断。为满足这种需要, 在 CPU 中设置一个“中断允许”触发器, 当 CPU 不允许中断时, 把“中断允许”触发器置“0”, 称“关中断”, CPU 允许中断时, 使用指令“开中断”, 把“允许中断”触发器置“1”。

② 有中断源请求中断。计算机可以处理哪些中断是在设计计算机时决定的, 我们把引起中断的原因称为中断源。因此 CPU 响应中断时, 必须有中断源请求中断。

③ 当前指令完成后才能响应中断。因为, 处理中断请求时, 还要保存 CPU 现场, 以便返回原程序。只有一条指令完成后才便于保存和返回现场, 因此又作此规定。

④ 申请中断的中断源的级别最高。因中断源的种类较多, 当它们同时请求中断时, 必须有一定的优先顺序, CPU 先响应级别最高的中断。另外数据通道和 DMA 的传送请求级别都比中断请求的级别高, 当响应某个中断源的请求时, 必须是当时它的优先级别最高。

⑤ 申请中断的中断源未被屏蔽。

设计机器时, 每一级中断的优先级都是规定了的, 是用硬件实现的。但有时希望改变中断源的优先级别, 可采用置“屏蔽”的办法, 把某个中断源的请求屏蔽起来, 这种中断提出申请也送不到 CPU, 因为这种中断源已经被屏蔽了。

具体屏蔽的办法是为每一个中断请求, 设置一个“中断屏蔽”触发器, 当该屏蔽触发器为“1”时, 表示该中断源不能申请中断。只有当该中断源的屏蔽触发器为“0”时, 该中断源才可请求中断。

置“中断屏蔽”与清除“中断屏蔽”都是用指令实现的。

**【例 1-108】** 什么是中断隐指令? 什么是中断周期? 中断周期完成什么工作?

解: 中断请求什么时候都可能出现, 但 CPU 只有在一条指令完成时, 才去检查有没有中断请求, 有关条件是否具备。如果条件具备, 则在此时即可响应中断, 转入中断处理程序, 此时有一些工作必须马上做, 如保存原程序断点, 转向中断程序入口等, 若用程序实现是很麻烦的, 因为不可能在每条指令后都设置这些指令, 另外也太耽误时间。为了响应中断, 实际上机器中都是采用硬件的办法, 设计一个硬件周期——中断周期。当有中断请求, CPU 允许中断, 在一条指令做完后进行检查, 一旦条件具备, 机器即转入“中断”周期, 采用硬件的办法完成紧要事项。不需响应中断时, 即不插入中断周期。

中断周期内完成三件事。

① 关中断

为了保存完整的 CPU 现场，转入中断保存现场时，不准再响应新的中断，直到现场保存完毕。

② 保存断点

断点是原程序停止时正在执行的指令的地址，为了处理完中断，返回原程序，必须把程序断点保存起来，以便继续执行原来的程序。

③ 转入中断处理程序总入口

中断周期做的事情相当于一条指令做的事情，但该指令在程序中又不出现，故有时又称为中断隐指令。

**【例 1-109】** 什么叫中断允许？什么叫中断屏蔽？为什么要设置中断允许与中断屏蔽？

解：CPU 执行程序有些时候不允许中断，特设立中断允许触发器，只有中断允许触发器为“1”才允许响应中断。如果禁止中断，可用指令将中断允许触发器置“0”，这时 CPU 就不再响应中断请求了。

中断源的优先级别是固定的，不能任意改变。为了控制各种设备中断的优先顺序，特设立中断屏蔽触发器，每一个中断源都对应的设立一个中断屏蔽触发器，当该屏蔽触发器为“1”时，其中断请求被屏蔽起来，不能向 CPU 申请中断，用这种方法可以改变许多设备同时请求中断时，CPU 响应哪个设备请求的次序。

**【例 1-110】** 说明中断处理过程。

解：CPU 响应中断，即进入中断周期，转入中断处理程序。

中断处理过程可分为三个阶段：

第一阶段，保存现场阶段。

- ① CPU 响应中断进入中断周期，保存断点，关中断
- ② 转入中断处理程序入口
- ③ 保存 CPU 现场寄存器内容
- ④ 进行中断排队，找出排上队并申请中断的中断源
- ⑤ 开中断

第二阶段，中断服务阶段。对于不同的中断源中断处理的方法是不同的，都有专门对应的中断服务程序。根据中断排队与识别，找出请求中断设备，用其设备编码作为该中断服务程序入口地址的一部分，转入其对应服务程序，完成规定的服务工作。

第三个阶段，恢复现场阶段。

- ① 关中断，在恢复现场阶段也不允许响应其他中断，打乱恢复现场的工作
- ② 恢复 CPU 现场寄存器内容
- ③ 开中断
- ④ 返回断点，返回原程序



### (5) DMA 控制方式例题分析

**【例 1-111】**为什么要设置“存储器直接访问”方式传送数据？说明 DMA 工作原理。

解：程序中断一定程度上解决 CPU 与外设、外设与外设并行工作问题，但每传送一个数据，CPU 都要响应一次中断，进入一次中断服务程序，服务前要保存 CPU 现场，服务后要恢复 CPU 现场，这都是用程序实现的，往往需要花几十条指令，花费不少时间，在外设增加时，中断服务将花费大量 CPU 时间，使 CPU 效率降低。

另外高速外设如磁盘、磁带，若采用中断方式传送数据，传送一个数据所需时间太长，可能会丢失数据，因此要求提供一种速度更快的数据传送方式。

中断方式耗费时间最多的是保存 CPU 现场和恢复 CPU 现场。因为执行中断处理程序时要使用 CPU 就会改变原来 CPU 的状况，而中断处理完毕又要返回原来的程序断点，继续执行原来的程序，又要恢复到原来的 CPU 的状况，因之产生了保存和恢复现场的要求。这都是通过读写主存的指令实现的。

缩短中断服务的时间，最好是不破坏 CPU 现场，也就不需要保存现场和恢复现场的工作。只要执行程序就要使用 CPU，就避免不了这个问题。如果传送数据不使用 CPU，就可简化这些步骤。

DMA 方式是“直接存储器访问”方式的简称，意思是以这种方式传送数据时是直接访问存储器来完成的，不经过 CPU，因此省去了保存和恢复现场问题。

访问存储器本来是由 CPU 执行访存指令实现的，现在不使用 CPU 怎么访存？这就要求专门构造一个 DMA 控制器，代替 CPU 执行访存功能，因此 DMA 控制器包括提供访存地址的地址寄存器，包括提供读写内存的命令，有的还包括交换数据的缓冲寄存器。这些都是用硬件实现的，就相当 CPU 访存时使用一个访存周期就可完成访存工作，DMA 方式访存时，使用一个 DMA 周期就可交换一个数据，速度很快，另外读盘读带都是一次交换一批数据，DMA 也可完成交换一批数据的要求，在 DMA 控制器中设置一个交换字数计数器，每交换一个数，字数 - 1，而地址 + 1；即可将下一个数写入到内存下一个地址中，直到字数计数器为 0 停止。

## 1.2.5 指令系统

指令是编程人员与计算机交互的最基本的手段，是机器指令的简称。指令包括操作码和地址码两部分，用二进制编码表示，因此才能把程序放到计算机中保存。

指令系统反映一台机器的主要性能。

### 1. 指令格式例题分析

一条指令的内容，应该包括该指令的操作运算种类和指定参加运算的操作数放在什么地方，有时还应该给出下条指令的地址，保证程序可以连续运行下去。

**【例 1-112】**指令字长应该如何决定？指令应该包括哪些内容？

解：计算程序是由一条一条指令组成的，指令和运算的数据一样放到存储器中保存，



需要时由存储器中一条一条取出来加以执行。因此,指令字长不是任意指定的,为了简化存储器的结构,充分发挥存储器中每一个存储单元的利用率,一般规定一个存储单元存放一条指令。为了照顾功能不同的长短指令的要求,一个存储单元可以存放多条指令,或一条指令存放在多个存储单元中。因为机器字长是字节长度的整数倍,指令字长也是字节长度的整数倍。

具体指令字的长度与指令包括的内容有关,如:操作码的位数,直接决定机器中指令的种类;地址码位数,决定可直接访问的存储器的容量;地址码的个数,直接影响地址码部分的长度,影响执行指令的具体操作。

**【例 1-113】** 什么叫一地址指令? 什么叫二地址指令? 什么叫三地址指令?

解: 一般指令中给出的信息数目越多,指令的功能越强。通常指令需要给出以下信息:

操作码 指明操作种类。

操作数地址 通常双操作数指令,如加减乘除指令需要给出两个操作数在存储器中存放的地址,有时还需要给出运算结果存放的地址。

下条指令地址。

但地址个数越多,指令字就越长,程序占主存的存储空间就越大。为了压缩指令字长,减少指令中地址个数是很有意义的。

#### (1) 三地址指令

指令中给出三个地址:两个操作数地址  $A_1$ 、 $A_2$ ,及运算结果存放地址  $A_3$ 。如果操作码用 OP 表示,则这种三地址指令执行的操作是:  $(A_1) OP (A_2) \rightarrow A_3$

其表示的含义是  $A_1$  单元的内容与  $A_2$  单元的内容执行 OP 规定的操作,运算结果放在  $A_3$  单元中。

关于下条指令的地址是这样解决的,程序中的指令是按照它们的执行顺序存放的,下条指令就放在本条指令之后那个单元中,本条指令的地址加 1,就是下条指令的地址。因此,一般机器中设置程序计数器 PC,指明本条指令地址,取出本条指令后  $(PC) + 1$  给出下条指令地址,这样在指令字中可以节省一个地址。

三地址指令地址数目较多,指令字较长,是其不便之处。

#### (2) 二地址指令

为了进一步减少指令中地址的个数,通常不给出运算结果存放的地址。这种指令格式规定运算结果存放到两个操作数地址中的一个地址中去。为了区分两个操作数地址,哪个地址存放运算结果,我们把既放操作数又放运算结果的地址叫目的操作数地址,另外一个地址称为源操作数地址。若源地址叫  $A_1$ ,目的地址叫  $A_2$ ,则其指令执行的操作是:  $(A_2) OP (A_1) \rightarrow A_2$ 。

二地址指令,地址数目较少,为不少机器采用,但指令执行结束时,目的地址中原来的操作数将被运算结果代替。

### (3) 一地址指令

为了进一步压缩指令字长,减少指令中地址个数,规定一条指令中,只给出一个操作数地址字段 A,另一个字段是操作码 OP。

对于双操作数指令,另一个操作数隐含的规定由运算器的累加寄存器提供。累加寄存器 AC 专门用于存放每条指令的运算结果;在运算器中是一个特别指定的惟一的寄存器。这种运算器又称单累加器结构。

一地址指令的操作含义是:  $(AC) OP(A) \rightarrow AC$ 。

累加寄存器 AC,不但提供一个操作数,而且存放每条指令的运算结果。

**【例 1-114】** 指令中操作码的位数是如何规定的?

解:指令中操作码的位数决定该机器中指令种类的多少。如大型机中,操作码可设置为 8 位二进制数(如 IBM 360/370),而微型机中操作码只设 4 位,但 4 位操作码,只有  $2^4 = 16$  种编码,可设置的机器指令最多只有 16 种,机器功能得太少。

为了增加指令的种类,又不增加指令字的长度,提出了一种可变长指令操作码方案,即操作码的位数对不同类型的指令长度不是固定的。对使用频繁的指令,采用短操作码方案,对不常使用的指令操作码位数可多一些,对于后者,当操作码位数增加时,地址码位数应该相应地减少。

不定长指令操作码方案,指令操作码编码灵活,指令种类限制较少,但操作码的译码控制要复杂一些。

### 2. 寻址方式例题分析

由于指令字长有限,地址码的位数有限,为了使指令能够访问更大的存储空间,指令中的地址码字段不直接给出要访问的操作数地址,而是只给出寻找操作数地址的位移量(形式地址)和寻找操作数地址的方法。根据位移量和寻找操作数地址方法的规定,找出操作数存放的实际地址(有效地址)。寻找操作数有效地址的方法叫寻址方式。

寻址方式不但可以扩大访问的存储空间,还可以为用户提供更加灵活的方法,使得对数组、循环指令的操作数访问更加方便。

**【例 1-115】** 什么叫直接寻址?什么叫间接寻址?

解:直接寻址方式是指指令中的地址码字段直接给出操作数的有效地址,不需再做什么地址变换,直接按地址码访问存储器,就可得到操作数。例如,操作码为 OP,地址码为 D,用直接寻址方式时,操作数有效地址  $EA = D$ ,地址单元 D 的内容,就是需要的操作数。

间接寻址方式,是指指令中地址码字段 D 给出的不是操作数的有效地址, D 单元的内容才是操作数的有效地址。也就是说,地址码字段 D 不是操作数有效地址,而是一个间接地址,按照地址 D 去访问存储器, D 单元的内容才是操作数的有效地址  $EA = (D)$ ,而地址码字段 D 称为间接地址  $IA = D$ ,间接地址单元的内容是有效地址  $EA = (IA)$

**【例 1-116】** 什么叫变址寻址?变址寻址有什么用途?某指令地址码字段为 8 位二

进制数, 变址寄存器为 16 位二进制数, 则指令可以访问的最大地址空间有多大?

解: 变址寻址方式中, 指令中地址码字段 D 给出的是一个位移量, 或叫形式地址, 另外还需指定一个寄存器作变址寄存器  $R_x$ , 变址寄存器的内容叫做变址量 ( $R_x$ ), 操作数的有效地址  $EA = (R_x) + D$

变址方式可以扩大访存空间, 当形式地址 D 是 8 位二进制数, 变址量 ( $R_x$ ) 是 16 位二进制数, 二者之和为 16 位二进制数, 因此访问主存的地址可由 8 位变成 16 位, CPU 访问主存空间可达  $2^{16} = 64K$  个单元。

另外一个用途, 当变址寄存器内容改变时, 访存地址将随着变址量的改变而改变, 而不需修改指令。有些机器指令系统中使用自增型变址和自减型变址, 就是每读一次变址寄存器, 其内容自动修改, “+1”或者“-1”, 因此对访问数组数据提供方便。

**【例 1-117】** 什么叫相对寻址? 说明相对寻址的特点。

解: 相对寻址是一种特殊的变址寻址, 当指定程序计数器 PC 作为变址寄存器使用时, 这种变址寻址叫做相对寻址。此时操作数的有效地址  $EA = (PC) + D$ 。

由于 PC 是程序指针, 存放本条指令地址, 每取一条指令,  $(PC) + 1$ , 给出下条指令地址, PC 的内容在运行程序时不断改变, 但操作数的有效地址是  $EA = (PC) + D$ , 也就是本条指令的操作数存放在从本条指令所在单元开始再数 D 个单元那个单元中。由于位移量 D 在指令中是一个常量, 因此操作数单元距本条指令单元间的距离是固定的, 相差 D 个单元。

相对寻址在程序搬家时很有用, 因为不管怎么搬家, 操作数单元距离本条指令单元的位置是固定不变的, 都是 D 个单元。

**【例 1-118】** 什么是寄存器寻址? 有什么特点?

解: 现代计算机运算器中存放数据的寄存器增多, 这样可以减少访问主存存数、取数的次数, 提高运算速度, 并且各寄存器在运算器中的地位和作用都是一样的, 不设专门的累加器, 每个数据寄存器都可作为累加器, 都可以存放运算结果, 这种运算器称为多累加器方案, 或通用寄存器结构, 为指令中运算带来很大方便。但必须给通用寄存器中每个寄存器一个编号, CPU 按寄存器编号从通用寄存器中取数或把结果写入到通用寄存器中。

这时机器指令地址码字段中给出的是寄存器编号, 执行指令时, 操作数可由通用寄存器读出, 不需访问存储器, 执行指令时间较快, 指令字长也较短。

采用寄存器寻址的指令中, 一般采用二地址方案, 即一条指令中给出两个寄存器地址 (寄存器编号), 一个叫源操作数寄存器, 另一个叫目的操作数寄存器, 目的寄存器除了在双操作数指令中提供一个操作数外, 还用来存放运算结果。

寄存器寻址方式特点是运算器采用通用寄存器方案, 如果运算器中有 16 个通用寄存器, 每个寄存器用 4 位二进制数进行编址, 则源寄存器和目的寄存器两个地址共需 8 位二进制数就够了, 寄存器地址位数较少, 因此指令字长较短, 取操作数不访问主存, 运算速度快。



仿照间接寻址,也可设寄存器间接寻址,此时寄存器的内容不是操作数,而是存放操作数的主存地址。当然此时取操作数仍要访问主存,但可减少取操作数有效地址的访问时间。

仿照直接寻址,寄存器寻址也可称为寄存器直接寻址。

【例 1-119】某机指令格式如下:

指令字长 16 位,高 6 位为操作码 OP 字段,低 8 位为形式地址,用补码表示,中间 2 位为寻址方式字段 X:

X = 00 表示直接寻址, X = 01 表示间接寻址,

X = 10 表示变址寻址, X = 11 表示相对寻址。

当程序计数器 (PC) = 1234H, 变址寄存器 ( $R_x$ ) = 0037H, 位移量 D = 20H, 主存单元 (0020H) = 1244H。

问:该机指令系统中最多可表示多少种指令?主存储器最大容量多少个字?求各种寻址方式下操作数的有效地址。

解:操作码 6 位,若采用固定长编码,指令系统中最多可设  $2^6 = 64$  种指令。

操作数地址最多 16 位,可访问的主存最大容量为  $2^{16} = 64K$  字。

各种寻址方式下的操作数有效地址分别是:①X = 00 直接寻址, D = 20H, 操作数有效地址 EA = D = 20H。

② X = 01, 间接寻址。

IA = D = 20H, EA = (IA) = (D) = (20H) = 1244H

③ X = 10, 变址寻址, 变址量 ( $R_x$ ) = 0037H, 位移量 D = 20H

有效地址 EA = ( $R_x$ ) + D = 0037H + 20H = 0057H

④ X = 11, 相对寻址。本条指令地址 (PC) = 1234H

位移量 D = 20H

有效地址 EA = (PC) + D = 1234H + 20H = 1254H

【例 1-120】说明堆栈寻址特点。

解:堆栈是主存中一个专门的存储区,其特点是访问堆栈时,不需要指定访问的单元地址,而是由堆栈指针自动给出访问单元的地址。例如出栈指令,指令中并不需要指定堆栈中读出哪个单元,只要指定是出栈操作,则堆栈自动把栈顶单元的内容弹出来送给运算器,具体的栈顶单元的地址是由堆栈指针给出的。

堆栈中设有栈底和堆栈指针 SP。堆栈空的时候,SP 指向栈底单元。当执行堆栈操作时,在向上生成的堆栈中,新写入的数据不能写入当前 SP 指出的栈顶单元中,而是先把堆栈指针加“1”,把将写入的数据写到栈顶单元上面一个单元中,此时 (SP) + 1 指出新写入数据的单元地址,即新装入数据的新栈顶单元。

出栈操作时,将堆栈指针指出的栈顶单元的内容弹出去送给运算器,注意:SP 永远指向写有数据的栈顶单元,当原来栈顶单元内容弹出后,则 (SP) - 1,指向原来写有数



据的新的栈顶。

因此堆栈寻址的特点,从栈底开始,先进栈的数据单元最接近栈底,以后进入堆栈的数据单元地址逐渐增大;而出栈时,先从最后进栈的数据单元向外弹出数据,每弹出一个数据堆栈指针(SP)-1,直到堆栈指针指向栈底,堆栈空了,就不能再执行出栈操作了。因此堆栈寻址的特点是先进入堆栈的数据最后才能读出,后进入堆栈的数据反而要先行读出。这种读出方法称为先进后出。在多重中断嵌套时很有用处。

### 3. 指令类型例题分析

**【例 1-121】** 分别说明字指令、字节指令、位处理指令、字符串指令的特点。

解:按照被处理的数据位数,指令可分为字指令、字节指令、位处理指令,字符串指令。

位处理指令是指对一个数据字中指定位进行处理,通常包括:按位置“1”,按位清“0”,按位测试等操作,一般都是逻辑运算。

字节处理指令是对一个字节 8 位数据进行处理,这时要求存储器提供字节编址的访问方式,一次可以读出一个指定的字节,多用于字符文档处理的应用中。

字处理指令,是指一次处理一个字长的数据。由于不同计算机中字长不同,字处理数据的位数也不同。如字长 32 位的机器,字指令一次读出 32 位二进制数,一次同时传送 32 位数,一次对 32 位数同时进行运算,一次把 32 位运算结果同时写入存储器中。

字符串指令,字符串常用于表示名称、记录、文本等。一个字符串占用连续的多个字节存储区域,字符串指令中要给出第一个字符的主存储单元地址和字符串长度。字符串指令包括字符串传送、比较、查找等操作。

指令的种类越多、机器的处理功能越强。

**【例 1-122】** 算术移位指令与逻辑移位指令有什么区别?

解:算术移位指令把被移位的数据作为一个带符号的数值来看,逻辑移位指令把操作数当作无符号数来对待。

主要区别在于右移操作时,填入最高位的数据不同。算术右移保持最高位(符号位)不变,而且符号位跟着向右移动,而逻辑右移最高位补“0”,向右移位时,最低位移入进位触发器中保存。

左移时、算术移位、逻辑移位最低位都是补“0”,最高位移入进位触发器中保存。

右移 1 位时相当于该数除以 2,左移 1 位相当于该数乘以 2。因此,用移位指令实现简单的乘除运算,可以提高运算速度。

## 1.2.6 计算机系统性评价

**【例 1-123】** 说明峰值 MIPS 与基准程序 MIPS 的意义。

解:MIPS 用来描述计算机的定点运算速度,表示当执行定点程序时,该机器每秒

钟能完成多少百万条指令。

对于一台计算机其峰值 MIPS 是以其指令集中基本指令的执行速度计算的。

可设一台计算机的基本指令执行时需要  $k$  个机器周期，每一机器周期时间为  $t \mu\text{s}$ 。

$$\text{则其峰值 MIPS} = \frac{1}{kt}$$

一台机器的平均 MIPS 是用其指令使用频度加权各类指令执行速度计算得到的。

基准程序 MIPS 值是用运行基准程序测得的 MIPS 值。由于不同的基准程序各种指令比例不同，因此在比较时必须说明是什么样的基准程序。

【例 1-124】说明平均故障间隔时间 MTBF 的意义。

解：故障间隔时间是指计算机运行中，两次相邻故障间的时间。平均故障间隔时间 MTBF 是多次故障间隔时间的平均值。它是衡量计算机系统可靠性的一个重要指标，MTBF 值越大，表示系统越可靠。

平均故障间隔时间，表示系统平均无故障时间也就是平均无差错时间。

如果系统运行一段较长的时间为  $t$ ，系统在这段时间内故障次数为  $N(t)$ 。

则系统的平均故障间隔时间

$$\text{MTBF} = \frac{t}{N(t)+1}$$

如果系统失效率是  $\lambda$ ，是包括各部件失效率之总和，则系统的平均故障时间

$$\text{MTBF} = \frac{1}{\lambda}$$

失效率可以通过各部件、元件的寿命实验获得，如果参加运行的元件总数是  $N$ ，元件失效前正常运行时间是  $T$ ，失效元件个数是  $n$

则其平均失效率

$$\bar{\lambda} = \frac{n}{NT}$$

【例 1-125】说明平均修复时间 MTTR 意义。

解：为了保证计算机系统处于正常工作状态或重新恢复到正常工作状态，必须采取的维护措施有检查、测试、调整、更换设备和修理等。其目的是尽量缩短系统停机时间，提高系统可用性。

计算机在使用过程中，通常包含检验、运行、故障、修改等工作构成一个周期。一个经过维护的系统其周期时间可包括：容许的最大可用时间，预防性的维护停机时间。随机出现的故障诊断时间和修复时间。

修复时间是从发现一次失效到恢复至正常状态所需的时间，它包括诊断、故障定位、修理等时间。平均修复时间 MTTR 是若干次修复时间的平均值。

平均修复时间是计算机系统可维性的重要技术指标，影响可维性因素有故障诊断设

计, 故障检测方法, 故障排除技术, 结构的可达性, 模块化设计, 可更换性等。

**【例 1-126】** 说明计算机系统可用性指标, 平均利用率的含义。

解: 可用性反映计算机系统在规定条件下正常工作的概率。一般用平均利用率 A 来表示。

$$A = \frac{MTBF}{MTBF + MTTR}$$

其中: MTBF 为系统平均故障间隔时间, 即平均无故障时间, MTTR 为平均修复时间, 即不能工作的平均故障时间, 则平均利用率也可表示为

$$A = \frac{\text{正常运行时间}}{\text{正常运行时间} + \text{故障时间}}$$

**【例 1-127】** 计算机安全在现代社会中越来越具有重要意义, 信息安全包括哪些内容信息安全的基本要素有哪些?

解: 计算机安全包括计算机系统资源安全和计算机信息资源安全。系统资源包括计算机硬件软件及有关设备。信息资源包括计算机中处理、存储、传送的大量的各种信息, 即数据资源。

信息安全的基本要素包括: 机密性、完整性、可用性(只有得到授权的实体才可访问数据)可控性(可以控制授权范围内信息的行为方式)及可审计性、能够对出现安全问题提供调查的依据和手段。

保障信息安全通常采用数据加密、数据备份、身份认证、入侵检测、防火墙等各种技术措施外, 还需特别强调安全管理, 划分安全等级, 制定相应管理制度, 以及制定相应的法律、道德规范等。

**【例 1-128】** 说明计算机病毒的特点。

解: 计算机病毒是在计算机程序中插入的一组破坏计算机功能和数据的程序代码具有很强的自我复制功能, 不断地扩大破坏的范围。

计算机病毒程序具有以下特点。

- ① 寄生性: 病毒程序不是独立存在的, 总是偷偷地附在其他文件上;
- ② 隐蔽性: 病毒程序进入系统总是偷偷地采用隐蔽方式进入系统, 当使用带有病毒的磁盘引导系统时, 病毒程序先进入主存再引导系统;
- ③ 非法性: 病毒执行的操作是未经授权的非法操作;
- ④ 传染性: 自我复制扩散;
- ⑤ 破坏性: 破坏操作环境、破坏系统资源、破坏文件、破坏数据。

**【例 1-129】** 说明防治计算机病毒的常用措施。

解: 病毒的入侵和反入侵的对抗是一场长期斗争。关键是加强对计算机系统的管理, 注意病毒入侵的预防措施。使用免疫软件和广谱的抗病毒软件及时检查病毒和杀灭病毒。

### 1.3 思考练习题及答案

#### 思考练习题

1. 某计算机字长 16 位浮点数格式:

0	1	5	6	15
Ms	E	M		

E 为阶码, 是 5 位二进制定点整数 (包括 1 位符号), Ms 为尾数符号占 1 位, M 为尾数的数值部分为二进制定点小数, 占 10 位。阶的底数为 2。求采用以下 5 种不同编码时, 十进制数  $-26.5$  表示成二进制浮点规格化数后的机器码。

阶码用补码, 尾数用原码, 机器码为 A。

阶码用补码, 尾数用补码, 机器码为 B。

阶码用移码, 尾数用原码, 机器码为 C。

阶码用移码, 尾数用补码, 机器码为 D。

A、B: ① 1001011101010000

② 1101011110101000

③ 1001010010110000

④ 1101010001011000

C、D: ① 1001011101010000

② 1101011101010000

③ 1101010010110000

④ 1001011110100000

2. 与十进制数 135.455078125 等值的十六进制数是 A。

某计算机字长为 8 位二进制数, 用原码、反码、补码、移码表示带符号的二进制整数 (最高位为符号位)。则机器码 11111111 代表的十进制真值分别是 B、C、D、E。

A: ① 87.351      ② 87.748      ③ 78.147      ④ 78.748

B~E: ① 128      ② 0      ③ 1      ④ -1

⑤ 255      ⑥ -255      ⑦ 127      ⑧ -127

3. 已知两个浮点数分别为  $X = 0.1101 \times 2^{010}$ ,  $Y = 0.1101 \times 2^{111}$ , 阶码为 3 位二进制补码。尾数为二进制补码, 对两浮点数求和。两数必须对阶。其阶差 (用十进制数表示) 为 A, 对阶后浮点数 Y 的尾数是 B, 相加后 C; 如果阶码用移码表示, 则 X 的阶码是 D, Y 的阶码是 E。

A: ① 4

② 3

③ 2

④ 1



- B: ① 1111                      ② 1.111                      ③ 11.11                      ④ 111.1  
      ⑤ 0.0001101                  ⑥ 0.01101
- C: ① 尾数无溢出                  ② 尾数有溢出                  ③ 尾数有进位                  ④ 阶码有溢出
- D、E: ① 110                      ② 010                      ③ 111                      ④ 011

4. 某计算机指令格式如下:

操作码	m	D
-----	---	---

其中 m 为寻址方式, D 为位移量。

寻址方式有多种:

直接寻址方式, 指令中的地址码是 A。

间接寻址方式, 指令中的地址码是 B。

变址寻址方式, 有效地址是 C。

堆栈指令是零地址指令, 访问堆栈单元的地址是由 D 决定的。

- A: ① 操作数                      ② 操作数地址  
      ③ 指令地址                      ④ 变址地址
- B: ① 操作数地址                      ② 操作数  
      ③ 存放操作数地址单元的地址                      ④ 变址地址
- C: ① D                      ② (D)                      ③ (PC)                      ④ (PC) + D
- D: ① PC                      ② SP                      ③ (PC) + D                      ④ (PC)

5. 某中断系统中, 中断源中断请求为  $I_1$ 、 $I_2$ 、 $I_3$ , 其中断优先级分别是一级、二级、三级, 优先级顺序由高到低。 $I_1$  优先级最高。中断请求  $I=0$ , 表示无中断请求;  $I=1$  表示有中断请求。 $M_1$ 、 $M_2$ 、 $M_3$  是分别对应  $I_1$ 、 $I_2$ 、 $I_3$  的中断屏蔽位,  $M=0$  不屏蔽中断请求,  $M=1$  屏蔽对应的中断请求。 $A_1$ 、 $A_2$ 、 $A_3$  分别是经过中断排队电路后  $I_1$ 、 $I_2$ 、 $I_3$  的输出信号,  $A_1$ 、 $A_2$ 、 $A_3$  的逻辑表达式分别为 A, B, C。

- A: ①  $I_1 M_1$                       ②  $I_1 \overline{M_1}$   
      ③  $\overline{I_2 M_2} \overline{I_3 M_3} I_1 \overline{M_1}$                       ④  $I_1 \overline{M_1} \overline{I_2} \overline{I_3}$
- B: ①  $\overline{I_1} M_1 I_2 \overline{M_2}$                       ②  $I_2 \overline{M_2}$   
      ③  $\overline{I_2 M_1} I_2 \overline{M_2}$                       ④  $I_1 \overline{M_2} \overline{I_1 M_1} I_2 \overline{M_2}$
- C: ①  $\overline{I_1 M_1} \overline{I_2 M_2} I_3 \overline{M_3}$                       ②  $I_3 \overline{M_3}$   
      ③  $\overline{I_1 M_1} \overline{I_2 M_2} I_3 \overline{M_3}$                       ④  $I_1 \overline{M_1} \overline{I_2 M_2} I_3 \overline{M_3}$

6. 某一双面磁盘, 每面 32 道, 每面盘格式化为 32 个扇区, 每个扇区包含 4 块数据, 每块数据 0.5KB, 则该盘容量为 A 字节。每个盘面都有一个磁头, 该盘能按要求作顺时针或逆时针旋转。各种操作时间执行如下所述。

旋转一圈时间为 320ms, 磁头从中心通过 32 个磁道到达边沿的时间为 32ms, 读一

块数据时间为 2ms，则该盘的平均等待时间接近 Bms，平均找道时间接近于 Cms。

设该盘上有 3 个文件都在同一盘面上。文件 X 在 6 磁道 2 扇区占 1 数据块；若磁头移动和盘转动不同时进行，磁头初始位置在 0 道 0 扇区，要读出文件 X 的时间接近于 Dms。

- A: ① 1.44M                      ② 2M                      ③ 4M                      ④ 4.096M  
 B~D: ① 10                      ② 12                      ③ 16                      ④ 28  
           ⑤ 40                      ⑥ 50                      ⑦ 60                      ⑧ 80

7. 选择正确答案:

- ① CPU 中执行逻辑运算都是在对应位间按位进行，各位之间没有进位关系。  
 ② 直接寻址方式中直接给出操作数。  
 ③ 闪存 FM 是随机读写存储器。  
 ④ SRAM 上的信息必须定时刷新，以便长期保存。  
 ⑤ DMA 方式中内存地址的提供和修改、传送字数的计数都是由 DMA 控制器硬件电路完成的。  
 ⑥ 常用的输入、输出方式有 3 种，即程序查询、程序中断、DMA 方式。  
 ⑦ 没有外部设备的计算机叫裸机。  
 ⑧ 微程序控制是多数计算机采用的控制方案，其主要优点是规整、维护、修改、扩充都很方便。  
 ⑨ 中断源在设计计算机时已经确定，用户不能随便增加。  
 ⑩ 硬盘的特点是速度快、容量大、价格便宜，是计算机的主存。

8. 计算机的存储系统一般由 3 级组成，即 cache、主存、辅助存储器。对 cache 最主要的要求是 A，cache 和主存一般都是 B 存储器。辅助存储器不能直接和 CPU 交换数据，对辅助存储器最主要的要求是 C。主存是计算机的主要存储器，CPU 可以直接访问它，又叫内存，当前主要采用 D 半导体芯片组成。

- A、C: ① 容量大                      ② 速度快                      ③ 价格低                      ④ 使用方便  
 B: ① 随机存取                      ② 顺序存取                      ③ 先进后出存取                      ④ 只读存取  
 D: ① ROM                      ② PROM                      ③ DRAM                      ④ SRAM

9. 将十进制数 563.6875 转换的八进制、十六进制数，再按多位分组法转换成二进制数。

10. 把下列二进制数转换成八进制、十六进制和十进制数。

- A: 1010    B: -101010    C: 10.0101    D: 101101.101

11. 写出下列各十进制数对应的二进制数的原码、补码和反码。

$$A = 6/16, B = 5/16, C = 1/16, D = 0, E = -1/16, F = -5/16, G = -7/16.$$

12. 已知数的原码，求该数补码。

- (A)<sub>原</sub> = 0.10101    (B)<sub>原</sub> = 1.10101    (C)<sub>原</sub> = 1.01011    (D)<sub>原</sub> = 0.01010

13. 已知数的补码, 求该数的原码。

(A)<sub>H</sub> = 0.1010 (B)<sub>H</sub> = 1.1010 (C)<sub>H</sub> = 1.1011 (D)<sub>H</sub> = 1.0011

14. 设一个 4 位二进制数  $X = 0.a_1a_2a_3a_4$ , 若  $X > 1/2$ ,  $a_1 \sim a_4$  取什么值。  $X \geq 1/8$ ,  $a_1 \sim a_4$  取什么值。

15. 设某机字长为 16 位二进制数, 问下列情况下其表示数的范围。

① 原码, 定点小数                  ② 补码, 定点小数

③ 原码, 定点整数                  ④ 补码, 定点整数

16.  $n$  位字长的定点二进制整数, 最高位为符号位。写出其补码, 反码情况下

① 最大正数                  ② 最小的负数                  ③  $-1$  的表示形式

④ 零的表示形式                  ⑤ 符号位的权

17. 用定点加减法求两个二进制定点小数  $X + Y$ ,  $X - Y$ , 并指出结果是否溢出。

$X = 0.11001$      $Y = 0.00111$

18. 采用补码运算主要为了\_\_\_\_\_。

19. 定点补码加减法运算时, 符号位\_\_\_\_\_运算, 运算结果的符号位为正确的结果符号位。

20. 定点数加减法运算可能产生溢出的情况是 A 和 B。

21. 补码加减法运算中采用双符号位目的是\_\_\_\_\_。

22. 定点除法中产生溢出的情况是\_\_\_\_\_。

23. 定点小数乘法\_\_\_\_\_产生溢出。

24. 浮点数的正、负符号是由\_\_\_\_\_决定的。

25. 浮点数运算可由 A 运算, 和 B 来实现。

26. 浮点数阶码运算, 只需完成\_\_\_\_\_运算就够了。

27. 浮点数判溢出, 是判\_\_\_\_\_溢出。

28. 浮点数尾数规格化时, 原码要求尾数最高数值位为\_\_\_\_\_。

29. 浮点数尾数规格化时, 补码要求尾数符号位与最高数值位\_\_\_\_\_。

30. 浮点数舍入, 恒置“1”法, 要求尾数末位\_\_\_\_\_。

31. 浮点数左规时, 尾数需要 A, 阶码需要 B。

32. 浮点数右规时, 尾数需要 A, 阶码需要 B。

33. 机器数中, 零的表示形式是惟一的是\_\_\_\_\_码。

34. 机器数中, 正数的符号用 1 表示的是\_\_\_\_\_码。

35. 机器数中, 负数的符号用“1”表示的是\_\_\_\_\_码。

36. 机器数中, \_\_\_\_\_码符号位负数用“0”表示。

37. 机器数中\_\_\_\_\_码符号位, 正数用“0”表示。

38. 定点小数中补码表示, 最小负数是\_\_\_\_\_。

39. 5 位定点二进制小数 (含 1 位符号) 用原码表示的最大正数是 A, 最小负数

是 B。

40. 主存用来存放 A 和 B，在程序运行时，直接供 CPU 存取。

41. 主机对主存的主要要求是 A、B 和 C、D。

42. CPU 可以直接按单元地址访问 A 和 B 存储器，不能按地址单元访问 C 和 D 存储器。

43. A 和 B 都是经常采用的半导体随机访问存储器，共同缺点是断电后 C 保存信息。

44. ROM 是一种只能读出不能写入的存储器，但其读出仍是 A，可以作为主存一部分使用。

45. 存储器地址是 10 位二进制数，其最大存储容量是 A，地址是 20 位二进制数，其最大存储容量是 B，地址是 30 位二进制数，其最大存储容量是 C。

46. 存储器带宽是指 A，提高带宽的方法是 B、C 和 D。

47. 随机访问存储器，要求访问任意存储单元的时间      与存储单元位置     。

48. MOS 半导体存储器具有如下特点 A、B、C，并且速度 D，可作大容量 E 使用。

49. ROM 是只读存储器，其工作特点是只能读出，不能写入。PROM 是 A 存储器，其工作特点是用户可以 B，工作时 C。

50. EPROM 是 A 存储器，其工作特点是 B，需要用 C 照射才能擦除其内容。工作时 D。

51. E<sup>2</sup>PROM 是电可擦除可编程只读存储器，擦除操作可以按 A 擦除和 B 擦除。

52. FM 称为闪存存储器，又叫 A，擦除工作是按 B 进行的，可以联机读写。工作原码与 C 相近，但速度很快。

53. 存储器中每一个存储单元存放 A，每个字单元有一个 B 编号，CPU 按 C 读写数据，每次读写 D 数据。

54. 主存储器组成包括 A、B、C、D、E，数据寄存器的位数与 F 一致。

55. 存储器芯片中采用行地址译码，列地址译码方案的好处是 A 和 B。

56. 片选信号用来 A 用，当片选信号  $\overline{CS}$  为高电位时，该芯片 B，当  $\overline{CS}$  为低电位该芯片 C。

57. 半导体存储器字扩展方式，扩展 A，位扩展方式扩展 B，位扩展时，各片数据线连接方法是 C。

58. 用 4K×1 位，RAM 芯片构成 4K×8 位存储器，需要 A 片 RAM，构成 16K×8 位存储器需要 B 片 RAM。

59. 双端口存储器的特点是 A，主要原因是一个存储器设置 B 电路。

60. cache 称为 A，它是为了解决 CPU 与主存 B 不匹配而采取的硬件措施。cache 的内容对用户是 C。



61. 虚拟存储器是为了解决主存 A 不够的矛盾, 通常使用 B 作辅助存储器使用。
62. 虚拟存储器中, 虚拟地址又叫 A, 主存地址又叫 B, 访问页式虚拟存储器中磁盘上的数据时, 必须先将磁盘上的程序和数据以 C 为单位调入主存, 访问主存时, 先通过查 D, 实现虚实地址转换, 最后才能访问主存, 取出有关数据, 这种地址转换是通过 E 实现的。
63. 计算机的各种存储部件中, 经常采用的有①主存、②cache、③硬盘、④通用寄存器、⑤光盘等, 其中 CPU 访问时速度最快的是 A, 容量最大的是 B, CPU 可按单元地址访问的存储器是 C, CPU 可以成批进行读写的存储器是 D。
64. 双总线结构计算机中的两个总线是 A 和 D。
65. 采用 Pentium CPU 的计算机中三级总线是 A、B 和 C。
66. PCI 总线是高速的 A 总线, 连接高速 I/O 设备, 它和 CPU 总线的工作是 B。
67. 许多部件都连接在一组总线上, 它们使用总线的方式是 A。
68. 多个部件同时要求使用总线发送数据时, 计算机必须设置总线 A, 控制总线的分配、使用和回收。
69. 总线上传送数据的通信方式可分为 A 和 B。
70. 收发双方按照统一的时间节拍发送和接收总线数据的通信方式称 A, 其特点是 B。
71. 收发双方每次传送数据都是通过请求和回答信号进行通信的称为 A, 其特点是 B。
72. 为了使不同主机厂家生产的机器与不同外设厂家生产的 I/O 设备能互连互换, 要求各个厂家都遵守通用 A, 其中包括: B、C、D、E。
73. IBM PC 总线又称 A 总线, B 位数据线, C 位地址线, 时钟频率 D, 共有引线 E 根。
74. ISA 总线, 又称 A 总线, B 位数据线, C 位地址线, 时钟频率 D, 共有引线 E 根。
75. EISA 总线, 又称 A 总线, B 位数据线, C 位地址线, 时钟频率 D。共有引线 E 根。
76. PCI 总线, 又称 A 总线, B 位数据/地址线, 时钟频率 C, 最高传输速率 D, 共有引线 E 根。
77. RS-232C 总线属于 A 总线, 常用于远程通信环境, 使用 B 作为通信介质, 引线数目 C 根。
78. USB 总线是一种通用 A 总线, 其特点是 B、C 和 D, 引线数目 E 根。
79. 程序查询方式传输数据时采用 A 方式, 此时 CPU 与 I/O 设备是 B 工作的。
80. 程序中断方式传输数据时采用 A 方式, 此时 CPU 与 I/O 设备是 B 工作的。

81. 中断方式传送数据是用 A 实现的, DMA 方式传送数据是用 B 实现的。
82. 中断方式传送数据执行中断程序时需要 A 和 B 以便返回原程序中断执行。
83. DMA 方式传送数据, 不经过 A, 不需要 B 和 C, 速度较快。
84. 中断方式一次传送 A 数据, DMA 方式一次传送 B 数据, 传送字节数目由 C 决定。
85. 中断方式主要用于 A 设备传送数据, DMA 方式主要用于 B 设备传送数据。
86. DMA 方式需要增加一个 A 控制器, 以便代替 B 控制总线, 实现 C 与 D 之间直接传送数据。
87. 一个计算机中设置 A 个“中断允许”触发器, 其主要作用是在特定情况下 B。
88. CPU 响应中断, 必须在 A 做完, 判断是否有人请求中断, 条件具备时, 立即转入 B。
89. 中断周期执行三件事 A、B 和 C, 是利用 D 实现的。
90. 中断处理程序开始时保留现场。  
保留现场开始时关中断, 其作用是 A。  
保留现场结束时开中断, 其作用是 B。
91. 中断嵌套原则是在处理低级中断过程中, 又产生高级中断, 则 A, 等高级中断处理完毕再继续处理低级中断。同级中断 B 互相打断。
92. DMA 传送数据开始前, 必须由 CPU 向 DMA 控制器预置 A、B 和 C, 才能开始传送工作。
93. 设备发出 DMA 传送请求, 必须等待 CPU A 结束后才能响应, 此时 CPU 把总线控制权交给 B, 传送一个数据需一个总线周期时间。
94. 串行通信主要用于 A 和 B 数据传输, 串行通信连线最少, 一次传送 C 位二进制数, 速度 D。
95. 计算机内部数据都是采用并行存放传送的, 串行通信数据都是串行传送的, 因此在串行通信接口内需设置 A 电路, 输出时需要把 B, 输入时需要把 C。

### 思考练习题答案

- |              |     |     |     |     |
|--------------|-----|-----|-----|-----|
| 1. A ①       | B ③ | C ② | D ③ |     |
| 2. A ②       | B ⑧ | C ② | D ④ | E ⑦ |
| 3. A ②       | B ⑤ | C ② | D ① | E ④ |
| 4. A ②       | B ③ | C ④ | D ② |     |
| 5. A ②       | B ③ | C ③ |     |     |
| 6. A ③       | B ⑧ | C ③ | D ④ |     |
| 7. 正确答案①⑤⑥⑧⑨ |     |     |     |     |
| 8. A ②       | B ① | C ① | D ③ |     |

9.  $(563.6875)_{10} = (1063.58)_8$   
 $(563.6875)_{10} = (233.B)_{16}$   
 $(563.6875)_{10} = (1000110011.1011)_2$
10. A:  $(1010)_2 = (12)_8 = (A)_{16} = (10)_{10}$   
 B:  $(-101010)_2 = (-52)_8 = -(2A)_{16} = (42)_{10}$   
 C:  $(10.0101)_2 = (2.24)_8 = (2.5)_{16} = (2.3125)_{10}$   
 D:  $(101101.101)_2 = (55.5)_8 = (2D.A)_{16} = (45.625)_{10}$
11.  $A = (6/16)_{10} = 0.0110$ ,  $(A)_{原} = 0.0110$ ,  $(A)_{补} = 0.0110$ ,  $(A)_{反} = 0.0110$   
 $B = (5/16)_{10} = 0.0101$ ,  $(B)_{原} = 0.0101$ ,  $(B)_{补} = 0.0110$ ,  $(B)_{反} = 0.0101$   
 $C = (1/10)_{10} = 0.0001$ ,  $(C)_{原} = 0.0001$ ,  $(C)_{补} = 0.0001$ ,  $(C)_{反} = 0.0001$   
 $D = 0 = 0.0000$ ,  $(D)_{原} = 0.0000$ ,  $(D)_{补} = 0.0000$ ,  $(D)_{反} = 0.0000$   
 $E = (-1/16)_{10} = -0.0001$ ,  $(E)_{原} = 1.0001$ ,  $(E)_{补} = 1.1111$ ,  $(E)_{反} = 1.1110$   
 $F = (-5/16)_{10} = -0.0101$ ,  $(F)_{原} = 1.0101$ ,  $(F)_{补} = 1.1011$ ,  $(F)_{反} = 1.1010$   
 $G = (-6/16)_{10} = -0.0110$ ,  $(G)_{原} = 1.0110$ ,  $(G)_{补} = 1.1010$ ,  $(G)_{反} = 1.1001$
12.  $(A)_{补} = 0.10101$   $(B)_{补} = 1.01011$   
 $(C)_{补} = 1.10101$   $(D)_{补} = 0.01010$
13.  $(A)_{原} = 0.1010$   $(B)_{原} = 1.0110$   
 $(C)_{原} = 1.0101$   $(D)_{原} = 1.1101$
14. 若  $x > 1/2$ , 则  $a_1 = 1$ ,  $a_2, a_3, a_4$  中至少有一个为 1。  
 若  $x \geq 1/8$ , 则  $a_3 = 1$ ,  $a_1, a_2, a_4$  任意。
15. 字长 16 位。
- ① 定点小数原码  $-(1 - 2^{-15}) \leq (A)_{原} \leq (1 - 2^{-15})$   
 ② 定点小数补码  $-1 \leq (A)_{补} \leq (1 - 2^{-15})$   
 ③ 定点整数原码  $-(2^{15} - 1) \leq (B)_{原} \leq (2^{15} - 1)$   
 ④ 定点整数补码  $-2^{15} \leq (B)_{补} \leq (2^{15} - 1)$
16.  $n$  位字长定点二进制整数, 最高位为符号位。
- ① 最大正数补码  $2^{n-1} - 1$ , 反码  $2^{n-1} - 1$   
 ② 最小负数补码  $-2^{n-1}$ , 反码  $-(2^{n-1} - 1)$   
 ③  $(-1)_{补} = (-0000000000000001)_{补} = 1111111111111111$   
 $(-1)_{反} = 1111111111111110$   
 ④  $0(\pm 0)_{补} = 0000000000000000$   
 $(+0)_{反} = 0000000000000000$   
 $(-0)_{反} = 1111111111111111$   
 ⑤ 符号位的权  $= 2^{15}$
17. 用双符号位法表示数的符号:

$x = 00.11001, y = 00.00111,$

$(x + y)_{\text{补}} = (x)_{\text{补}} + (y)_{\text{补}} = 01.00000$  溢出

$(x - y)_{\text{补}} = (x)_{\text{补}} + (-y)_{\text{补}} = 00.10010$  不溢出

18. 把减法运算变成加法运算

19. 参加

20. A. 同号两数相加

B. 异号两数相减

21. 判断溢出

22. 被除数绝对值大于除数绝对值

23. 不会

24. 尾数符号

25. A. 阶码

B. 尾数

26. 加减法

27. 阶码

28. 1

29. 不同

30. 置“1”

31. A. 尾数左移 1 位,

B. 阶码 - 1

32. A. 尾数右移 1 位,

B. 阶码 + 1

33. 补码、移码

34. 移码

35. 原码、反码、补码

36. 移码

37. 原码、反码、补码

38. -1

39. A.  $0.1111 = +\frac{15}{16}$

B.  $-0.111 = -\frac{15}{16}$

40. A. 程序

B. 数据

41. A. 速度快

B. 容量大

C. 成本低

D. 可靠

42. A. RAM

B. ROM

C. 磁盘

D. 磁带

43. A. SRAM

B. DRAM

C. 不能

44. A. 随机

45. A. 1K

B. 1M

C. 1G

46. A. 一秒钟读出的二进制位数

B. 减小存取周期时间

C. 增加字长

D. 增加存储体数目

47. A. 相等

B. 无关



48. A. 集成度高 B. 功耗低 C. 价格低 D. 较快  
E. 随机访问存储器
49. A. 可编程只读 B. 用户进行一次编程 C. 只读不写
50. A. 可改写可编程只读存储器 B. 可擦除原来存储内容  
C. 紫外光 D. 只读不写
51. A. 字节 B. 全部
52. A. 快擦存储器 B. 数据块 C. E<sup>2</sup>PROM
53. A. 一个数据字 B. 地址 C. 地址 D. 一个字长
54. A. 存储体 B. 地址寄存器 C. 地址译码器 D. 数据寄存器  
E. 读写控制电路 F. 字长
55. A. 节省芯片引出脚数目 B. 节省译码电路
56. A. 扩充容量 B. 停止工作 C. 允许访问
57. A. 存储容量 B. 字长  
C. 单独引出, 连接数据总线
58. A. 8 B. 32
59. A. 存取速度较快 B. 两套单独的读写电路
60. A. 高速缓冲存储器 B. 速度 C. 透明的
61. A. 容量 B. 磁盘
62. A. 逻辑地址 B. 物理地址  
C. 页 D. 查页表 E. 软件方法
63. A. ④通用寄存器 B. ⑤光盘  
C. ①主存 D. ③硬盘
64. A. 存储总线 B. I/O 总线
65. A. CPU 总线 B. PCI 总线 C. ISA 总线
66. A. 同步总线 B. 异步的
67. A. 分时共享
68. A. 仲裁机构
69. A. 同步方式 B. 异步方式
70. A. 同步方式 B. 控制简单, 速度快
71. A. 异步方式 B. 控制复杂, 速度慢, 可靠
72. A. 总线标准 B. 功能规范  
C. 信号电平规范 D. 插头座尺寸、定位 E. 时序规范
73. A. XT 总线 B. 8  
C. 20 D. 4.77MHz E. 62
74. A. 工业标准化总线 B. 16

- C. 24  
75. A. 扩展的工业标准总线  
C. 33  
76. A. 外围设备互联总线  
C. 33MHz  
77. A. 异步  
78. A. 串行  
C. 即插即用  
79. A. 异步  
80. A. 异步  
81. A. 程序  
82. A. 保存现场  
83. A. CPU  
84. A. 一个字节  
85. A. 低速  
86. A. DMA  
C. I/O 设备  
87. A. 一  
88. A. 一条指令  
89. A. 关中断  
C. 转入中断处理程序入口地址  
90. A. 禁止响应新中断  
91. A. 高级中断可以打断正在进行的低级中断服务程序  
B. 不能  
92. A. 开始传送的内存首地址  
C. DMA 传送一个数块的字节数目  
93. A. 一个总线周期  
94. A. 远程  
C. 1  
95. A. 串/并行转换  
C. 串行数据变成并行数据
- D. 8MHz  
B. 32  
D. 8.33MHz  
B. 32/64  
D. 132/264  
B. 电话线  
B. 高速  
D. 容易扩充  
B. 串行  
B. 并行  
B. 硬件  
B. 恢复现场  
B. 保存现场  
B. 多个字节  
B. 高速  
B. CPU  
D. 主存储器  
B. 不允许响应新的中断请求  
B. 中断周期  
B. 保存断点  
D. 硬件  
B. 允许响应所产生的高级中断请求  
B. 读/写命令  
B. DMA 控制器  
B. 低速设备  
D. 较慢  
B. 并行数据变成串行数据
- E.  $62 + 36$   
E. 196  
E. 45  
C. 25  
E. 4  
C. 恢复现场  
C. 字节数计数器

## 第2章 操作系统基础知识

### 2.1 内容提要

操作系统是一个大型的软件系统，是系统软件最主要部分。在整个计算机系统中，操作系统位于所有软件的最底层，直接与计算机硬件相衔接。它的主要任务是管理系统的资源，包括硬资源和软资源。所谓硬资源是指 CPU 资源、各类寄存器资源、内存资源、各种外部设备资源等，所谓软资源，是指文件、用户资料与作业资料以及其他有关的驱动程序、模板等。通过操作系统的管理，可以屏蔽使用这些资源时涉及的硬件细节，提高用户使用的抽象级别，使用户可以更加方便、简捷地使用资源。

操作系统的目标是使系统资源得到有效、合理和方便的使用。所谓有效，就是提高资源的利用率，这可以通过在操作系统的调度下，多个用户按照一定的次序交替使用同一个资源，借以提高资源的利用率；所谓合理，就是在通过多用户共享来提高资源利用率的同时必须制定一系列规则，保证每个用户的合法使用权，防止某个用户在不可预见的长时间内不能使用资源；所谓方便，就是提高用户使用的抽象级别，尽可能替用户分担繁杂的工作，同时在用户界面上尽可能地友好。

操作系统的基本特征是多任务并行和多用户资源共享。前者是手段，后者是目的。

观察与分析操作系统可以从不同的观点出发。所谓用户观点是从系统提供的功能和服务出发，即根据它提供的系统调用和用户联机命令来研究操作系统。所谓资源管理观点是从系统管理的资源类别出发，将操作系统分为处理机管理、存储管理、设备管理、文件管理、用户与作业管理等几个大部分。所谓进程观点是从系统运行的特点出发，所有服务和管理都是通过进程的运行来实现的，这样系统可以分为若干个进程和一个对进程进行管理与协调的系统内核。所谓分层观点是从操作系统构造的角度出发，将系统分为若干个层次，彼此之间存在支持和依赖关系。

通常，操作系统的学习与课程讲授在内容上从资源管理的观点来组织，即大致分为处理机管理、存储管理、文件管理、设备管理和作业管理几个部分，分别讲授管理的对象、内容以及涉及的各种管理策略与算法；而从操作系统运行的角度，则采用进程观点来讲述操作系统对并行问题的组织与处理，这部分内容主要集中在操作系统内核部分，结合处理机管理来讲授。

学习操作系统的主要难点在于如何理解操作系统所体现的并行行为及其实现机理。操作系统的基本特征是并行与共享，或者说多任务并行与多用户资源共享。其中，共享是目的，通过多用户的资源共享，提高系统的资源利用率；并行则是手段，只有实现

多任务并行,即将一个物理的处理机(CPU)转化为多个虚处理机(进程处理机),使多个任务同时运行,交叉地使用系统的资源,避免一种资源长期为一个用户所独占,达到资源充分利用的目的。

除了并行问题以外,操作系统中管理资源的方法,包括资源的组织、分配算法,以及实现资源之间相互转化等问题也是操作系统的难点。需要说明的是,通过操作系统的管理,资源之间是可以相互转化的。例如,通过内外存交换,就可以实现外存资源加上CPU资源与内存资源的转化,也就是说,在用户看来所有数据似乎都放在内存,占用的都是内存资源,但实现上却只在内存放置部分数据,在使用时通过CPU进行内外存之间交换的方式来实现所有数据的访问。

我们试图通过对典型的例题进行分析与讲授的方式来对操作系统的重点内容和难点问题加以阐述。通过对典型问题的讲述,使读者能够对该方面的知识有一个比较全面地了解,并掌握其中的主要内容与方法。

### 2.1.1 操作系统内核与处理机管理

操作系统核心的主要内容是处理机(CPU)管理。核心的主要功能是通过软件(当然是在硬件的基础上)实现处理机的扩充,将一个物理的CPU扩充为多个虚CPU(也称进程虚拟机),实现多个进程的并行运行。通过核心程序的运行,向上呈现出提供若干个无中断概念的进程虚拟机。

#### 1. 程序与进程

程序是一个静态的概念。所谓程序,就是指令或语句的有序集合,它实现了某一个算法,它可以独立地存在。通过程序的运行,可以得到根据算法与初始条件的计算结果。由于冯·诺依曼计算机的特点,所有的程序都是顺序执行的。单个程序的运行过程是稳定的和可以预期的。而进程是一个动态的概念。所谓进程,是指程序的一次执行,进程的主体部分是程序,但是,进程必须与操作系统密切相关,即与进程管理、处理机调度等结合在一起,才能反映出进程的行为。从这个意义上,进程是一个静态定义和动态运行相结合的产物,离开了操作系统的管理机制,单独对进程下定义是没有意义的。当然,进程本身除了程序以外,还包括数据和进程控制块。

#### 2. 操作系统的硬件基础

操作系统的硬件基础主要包括中断与通道两大部分。中断是操作系统实现多个程序并行的基础,在系统中处于枢纽的地位。用户需要操作系统提供的任何服务都会导致中断的出现,由用户程序的运行切换到操作系统程序运行。因此,任何操作系统程序的运行都是由中断来驱动的,操作系统的入口地址就是中断响应后计算机转而运行的地址。

通道则是实现计算与外部设备运行的基础,操作系统中的并行主要体现在计算与设备的并行和计算与计算的并行。这里,由于通道可以独立于CPU控制设备的工作,因此



设备与计算可以在物理上实现并行工作；由于系统中只有一个 CPU，计算机在同一时间瞬间只能执行一条指令，计算的并行是不能在微观上实现的，所谓计算之间的并行，只能是在宏观上实现，即系统中存在多个进程，它们的运行呈现此起彼伏的现象，进程走走停停，CPU 在进程之间切换，忽而运行这个进程，忽而运行另一个进程。

### 3. 多道程序的工作原理

多道程序运行是现代操作系统的主要特点之一。所谓多道程序，是指计算机中有多个程序在同时运行。在单 CPU 情形，不可能在某一瞬间同时执行多个程序。所谓多个程序同时运行，是指 CPU 交替地执行多个程序，而不是一个程序执行结束后再执行另一个程序。这意味着在一个程序执行结束之前，另一个程序可以开始其执行。操作系统正是通过这种手法来实现多个任务的并行的。例如，某个程序执行过程中，需要启动外部设备进行输入输出，这时，该程序启动了外设控制设备（例如通道），外设在通道的控制下工作，而程序必须等待输入输出的完成才可以继续运行，操作系统此时就调度另一个程序运行，实现了两个程序的并行。

### 4. 进程的概念

进程是一个动态的概念。通常称进程为程序的一次执行。进程的主体部分是程序，但是与程序不同，进程具有状态，它有一个进程控制块，进程控制块是进程存在的惟一标志。而且，进程的行为也与程序不同，进程具有状态，即运行、就绪和等待三种状态，操作系统分别设置不同的队列来指示处于不同状态的进程在某个队列中的先后次序。只有进程处于运行态时，它的运行才体现出程序的行为。进程在这三种状态之间切换，这时，CPU 相应地就在不同进程之间切换运行它们的程序。

因此，进程是一个软硬件结合的概念。通过操作系统 CPU 管理程序的运行，多个进程呈现出并行运行的行为。这里，进程行为的体现不仅依赖于进程自身程序的逻辑，也依赖于中断机制，甚至包括中断到来的时机，还依赖于操作系统进程调度的算法等。

### 5. 进程之间的通信

进程之间的通信，主要是为了建立正确地实现进程之间的数据交换机制。这里不讨论数据交换的具体方式，而是把目标集中在正确实现数据交换过程中进程之间关系，主要是同步和互斥关系的处理问题。确切地说，进程之间的通信有两种实现机制，即利用共享的数据结构进行通信和消息传送。正确的通信机制必须要正确处理进程之间的制约关系，只要保证直接制约关系的正确性，相对制约关系的正确性就自然得到保证了。

### 6. 同步问题和信号量与 P、V 操作

进程之间的通信应该防止出现与时间有关的错误，所谓与时间有关的错误是指这类错误是很难重复出现的。这类错误的起因是因为进程之间通信时，相互关系处理不正确。而错误的表现形式不仅与关系不正确有关，而且与进程运行的环境有关。互斥是一种特殊的同步。进程之间的同步不能由进程自己来解决，必须通过系统核心加以控制与协调。信号量是一种核心内的数据结构，在信号量上只能施行两种操作，即 P、V 操作。

## 7. 进程管理

进程由核心进行管理, 进程控制块是核心中最重要的数据结构, 是进程调度的依据。进程管理主要包括现场管理、队列管理、进程调度等。现场管理负责将被中断打断运行的进程运行现场保留起来, 也负责将被保留的现场加以恢复, 是被打断的进程恢复运行; 队列管理进程可以由核心创建和撤销, 一个进程也可以要求核心为其生成一个子进程, 从而形成进程家族, 父、子进程共享资源。

## 8. 中断响应

中断响应是一个软硬件结合起来处理系统例外事件的机制。硬件响应中断的工作时进行新老程序状态字的交换。所谓程序状态字, 是指 CPU 一些重要寄存器内容的有序集合。老程序状态字是指系统正在运行时的程序状态字, 新程序状态字是指存放在内存指定单元的程序状态字, 新程序状态字中的指令地址寄存器内容就是操作系统的入口地址。通过交换程序状态字, 系统转入运行操作系统的程序。

接下来, 中断响应的工作将由操作系统来完成。操作系统判别产生中断的原因, 根据中断的原因调用相应的中断处理例程, 完成中断的处理。在中断处理结束以后, 再运行进程管理中的进程调度程序, 将某个进程运行时的程序状态字内容填入相应的硬件寄存器中, 从而使该进程投入运行。新投入运行的进程既可以是原来被打断的进程, 也可以是另一个进程。

### 2.1.2 存储管理

存储管理包括内存管理和虚拟存储两个大部分。内存管理的主要工作是如何在不同用户、不同进程之间分配与回收内存资源, 虚拟存储则是如何实现进程运行时的内外存交换, 从而使进程获得比它所使用的内存空间更大的编址空间, 亦即进程可以访问的空间将可以大于它所获得的内存空间。内存管理将涉及内存空间的组织、分配与管理策略, 也将直接决定程序中所使用的相对地址形式。虚拟存储机制将设计工作方式、淘汰算法以及改善系统性能的技巧等。

#### 1. 地址重定位

所谓地址重定位, 是指将用户程序中出现的逻辑地址 (又称相对地址) 转化为计算机可以直接寻址的物理地址 (又称绝对地址) 的过程。由于在系统中可能由多个用户程序存在并且并发运行, 而且用户程序是通过操作系统进入内存并由操作系统加以管理, 程序的编制者和编译程序根本无法预知操作系统会将程序及其运行数据存放在什么地方, 更何况目标代码需要经过连接装配才变为可执行代码。编译程序只能从零开始安排数据区和程序区, 直到操作系统为其分配存放区域后才将地址映射到实存地址。

程序运行前完成全部重定位过程称为静态重定位, 在程序运行过程中就出现的操作数逐个进行重定位称为动态重定位。动态重定位工作是由软硬件共同实现的, 操作系统准备与提供定位依据 (数据), 硬件完成定位操作。

## 2. 实存管理

内存管理策略包括固定分区、段式管理、页式管理、段页式管理几种。所谓固定分区是一种静态管理策略，将内存分为若干个长度固定、用途固定的分区，按内容分别存放；段式管理采用内存动态分割、段内连续分配的方式，根据用户要求动态分配和释放内存；页式管理采用内存固定分割、根据用户要求以页为单位分配与释放内存；段页式管理则结合段式管理和页式管理的特点，采用内存固定分割为页、离散页组合成段、以段为单位申请与释放内存；这样在用户层面呈现段式管理的特性，而在系统管理与内存分割上又呈现页式管理的特性，保留各自的特点与优点。

内存管理的实现是由软、硬件共同完成的，各种管理都有特定的硬件机制支持。

## 3. 虚存管理

虚存管理是一种典型的由操作系统实现资源转化的例子。通过操作系统的虚存管理，CPU 资源和外存资源可以转化为内存资源，从而扩大进程的编址空间。利用程序的局部性特点，采用内外存交换技术，可以向用户提供比物理内存更大的虚存空间。

由于在虚存系统中，进程计算时的操作对象只有部分存放在内存（在外存也有副本），相当部分操作对象存放在外存，因此进程运行中将会发生需要访问的数据不在内存的情形。这时，系统将发出“缺页”中断，转入操作系统程序运行。操作系统首先要将内存中的一个页面淘汰出去，将其放回外存，然后再将所需要的操作对象所在页面调入内存，占据刚刚淘汰出去页面所占据的位置。这样，进程就可以继续运行了。

虚存最突出的问题是性能问题。淘汰算法的选择、程序设计技巧可以部分地有助于性能的改善，但最根本的问题还是实存空间与虚存空间的比例，在这个意义上，虚存空间是不可能无限增大的。

## 2.1.3 文件管理

文件是操作系统管理的一类重要资源，属于软资源的范畴。现代操作系统基本上不再规定文件的内部结构，而采用流式文件机制。当然，文件式可以有其内部结构的，现代操作系统将文件的内部结构放在用户层面来解决。而从操作系统角度将文件看成一个线性的内容串。它的内部结构由用户自己在使用时处理。

### 1. 文件的结构与组织

文件是存储在辅助存储器（通常是磁盘，后备文件也存储在磁带上）中的具有标识的一组信息集合。对文件的分类可以从各种不同的角度。从文件的内部的信息组织结构来分，文件的逻辑结构可以分为记录文件、流式文件等。实际上，这就是文件有结构和无结构的区别。记录文件内，文件的内容被分成若干个纪录，每个记录具有相同的结构，记录是访问文件的最小单位，记录有记录名，可以按名字访问。而流式文件则没有记录的概念，整个文件被看成是一个大的串，相接成一片，组成串的最小单位可以是字符，这时称为字符串文件；也可以是二进制数，这时称为二进制文件。



从文件内容的存放与查找方式来分,文件的物理结构可以分为连续文件、直接文件、链接文件、索引文件、索引顺序文件等。连续文件的内容按先后次序存放在连续的磁盘空间内。直接文件通过散列算法建立文件控制块在目录中的存放地址(用序号表示)与其名字的对应关系。链接文件则将文件的各部分内容组织成链表的形式,各部分通过指针链接。可以使用单链接,这时指针在上一部分,指向文件的下一部分;也可以使用双链接,这时上下两个部分都有指针,各自指向对方。索引文件则配备一张索引表,建立文件各部分(通常以页号或记录号来表示)与其存放的磁盘地址的对应关系。索引顺序文件与索引文件类似,但规定磁盘地址不能随意,地址大小要与内容先后一致起来。

## 2. 文件系统

文件系统是文件及其管理机构的总称。每个文件都有一个文件控制块,它是文件存在的惟一标志。为了方便文件的检索,应该将文件控制块按照某种规则有序地放置,文件控制块的有序集合称为文件目录。

文件目录的组织形式包括多种。一级目录,这时所有的文件控制块被组织成一张顺序表的形式,这种组织形式虽然简单,但查找文件效率就比较低,例如文件系统中有100个文件主,每个文件主拥有100个文件,总共有10 000个文件,平均查找长度就为5 000。

二级目录,将系统中的文件按某种规则先组织成多张表,再按表的类别组织成一张表,形成二级表的组织形式。例如在上例中,先按文件拥有者组织文件,形成100张表(每表100个文件),再将所有的文件主组织成一张表(有100个表项),将文件主与其对应的文件表地址对应起来。查找文件时,需给出文件拥有者与文件名两个信息。首先,按文件拥有者找到相应文件表的起始地址(平均查找长度为50),再在相应的文件表中找出所要的文件(平均查找长度也是50),总的查找长度为100,比一级目录快多了。

多级目录,通常为树型目录。这时,系统将设立几级中间目录,文件是树型目录中的叶结点,中间结点(包括根结点)都是目录文件。每个目录文件是一张顺序表,表中依次列出它的下级目录名及其地址。一个文件的查找需要使用从根结点开始的路径名。在上例中,我们设置五级目录,每个上级结点下辖10个下级结点。这样,每级的查找长度为5,查找一个文件的平均长度为25,查找速度更加快了。

同时,文件系统要为用户提供多种使用文件的手段。

## 3. 磁盘空间管理

文件通常是存放在磁盘上的,当创建文件时,需要申请空闲的磁盘区域来存放其信息,当删除文件时,需要回收其占用的磁盘区域。通常,磁盘空间被划分为磁盘块,物理磁盘块的查找地址形式是一个三元组(柱面号,磁道号,块号)。物理上,磁盘由若干个上下同轴的磁盘片于相应的读写磁头组成,每个磁盘上有若干个同心圆形状的磁道,磁道上可以记录数据。每个磁道又被分为若干个区域(区域之间有间隔),磁头可以沿半



径方向移动,对准磁道。不同磁盘片上相同磁道号的同心圆组成一个圆柱面,可以根据柱面号加以区分。因此,给定柱面号就惟一确定了磁盘块所在的同心圆大小。磁盘片按由上至下编号(如是双面磁盘,则一个磁盘片有相邻的两个编号)三元组中的磁道号就代表这个编号。这样,给定了柱面号和磁道号,就惟一确定了磁盘块所在的磁盘面以及所在的同心圆。块号则是同一同心圆上磁盘块划分的序号,给定了三元组,就惟一确定了该磁盘块的位置了。为了叙述简单,通常用一个数字来描述一个三元组,显然这两者是等价的。

磁盘空间的管理主要体现磁盘空闲块的管理上,通常有位图(页示图)法、链接法、索引法和成组空块链接法等。它们的共同之处是都要记录磁盘块的使用状况以及地址,设计与实现分配操作与回收操作;不同之处则是记录的格式不同、存放的方式不同,由此导致实现分配和回收的方法也不同。

#### 4. 文件的使用

文件系统提供了一系列使用文件的手段,包括打开文件、关闭文件、改变文件系统的当前目录(或使用目录)、读文件、写文件、文件定向等。

需要说明的是,文件系统为每个用户提供一张打开文件表,记录该用户打开的文件,每个文件占据一个表栏目。因此,栏目序号就可以用来代表这个已经被打开的文件,每打开一个文件,系统将返回一个表栏目号(称打开文件号),此后的文件操作就使用打开文件号。当一个用户建立时,系统自动为它打开三个文件,序号为 0, 1, 2。分别表示键盘(或鼠标)输入文件、屏幕输出文件以及错误信息文件。也就是说,用户生成时,系统默认为其配置了一台屏幕输出设备和一台键盘(鼠标)输入设备。通过替换打开文件号,就可以容易地实现输入、输出的重定位。

### 2.1.4 设备管理

设备是操作系统管理的重要硬件资源。实际上,操作系统程序的一大部分是用来管理设备和驱动设备的。但是,设备的使用与文件的使用在形式上非常近似。例如,输入设备向内存传送数据,就可以类比为从文件读取数据;输出设备将内存中的数据以硬拷贝的形式输出,可以类比为向文件写出数据。因此,现代操作系统往往将设备管理并入文件管理部分。当然,对设备的处理与对文件的处理是不一样的,这种合并只是形式上的合并,其操作语义的处理还需另外对待。

#### 1. 设备的类型

设备的类型划分直接依赖于分类的观点。从设备分配与使用的观点来看,设备可以分为两类:共享设备与独占设备。共享设备的使用权可以比较容易地在用户之间转移,不需进行其他额外的工作。独占设备在一个程序运行期间适宜归其独占使用,如果在此期间使用权发生转移,将会带来许多不便之处。

而从设备与计算机数据交换的观点来看,设备又可以分为两类:字符设备与块设备。

字符设备与主机交换数据的单位是字符，因此速度较慢，又称慢速设备，如终端、打印机、网络接口等；块设备与主机交换数据的单位是块，速度较快，也称快速设备，如磁盘、磁带、闪存设备等。

通过操作系统的管理，还可以实现计算机资源的转换，也可以实现设备类型的转换。由此还引出了虚设备，所谓虚设备技术，就是用一类设备（通常是快速设备）来模拟另一类设备（通常是低速设备）的技术，当然设备也可以模拟自身，例如多窗口技术就是屏幕输出设备的自身模拟，一台变成了多台。被模拟的设备就成为虚设备。例如，用一个文件来虚拟一个打印机，将程序在运行过程中的零星输出转向存入文件，待程序运行结束后一并输出，一个文件就模拟了一台打印机，独占设备变成了共享设备。

## 2. 设备的控制、连接与管理

外部设备完成输入/输出任务，在计算机的控制下进行。但是计算机对设备的控制有多种方式，如直接方式、通道方式、I/O 处理机方式、DMA（控制器）方式等。其中，直接方式是早期使用的设备控制方式，计算机直接使用程序控制设备的动作，为输出设备从主存中提取数据，为输入设备向主存存入数据，由于 CPU 与设备运行存在巨大的速度差异，计算机经常处于等待状态，计算资源很难得到充分利用。

通道方式利用通道（一类特殊的专管 I/O 处理的专用处理机）控制外部设备，通道可以执行通道程序（专司输入输出的指令串）直接从事设备与内存交换数据，并控制设备工作，CPU 只需为通道准备好通道程序，向通道发出启动指令，通道便可以自行控制设备完成 I/O 工作，CPU 可以转而运行别的程序。工作完成（或出现异常情况时）后，发出中断，提醒主机进行结束处理或者例外事件处理。一般大型计算机配备通道。

I/O 处理机的工作方式与通道类似，但没有像通道那样具有完全的 I/O 操作独立性。它可以独立控制设备完成数据的输入或输出，但是设备只能与 I/O 模块内置的缓冲区进行数据交换。例如，当输入数据完成后，I/O 模块需要发出中断提醒 CPU，由 CPU 完成缓冲区与内存的数据交换。显然，它较之直接控制方式大大减轻了主机的压力，但比通道还需要更多的主机干预。

DMA 方式与通道方式类似，DMA 部件可以直接运行 I/O 程序，控制设备与内存以字节为单位交换数据。但是，DMA 模块与主机共享数据总线周期，因此，在 DMA 传送数据的过程中，CPU 访问总线的速度会放慢。

目前，微机内大多采用 I/O 处理部件或 DMA 方式。

由于用户对设备的使用方式与文件使用方式类似，现代计算机操作系统把对设备的管理放在文件系统中。

### 2.1.5 作业管理与用户界面

作业是操作系统管理的又一类软资源。作业可以分为联机与脱机两类，它们的组织与管理也完全不一样。操作系统的用户界面是指它提供服务的方式，操作系统向用户提

供三种不同类型的服务：命令服务、系统调用服务和作业控制语言服务。它们分别针对联机用户、程序运行和脱机用户。至于屏幕操作与显示的差别，只是表现形式的不同，没有语义上和原理上的差别。

### 1. 用户作业管理

用户作业一般分为两类：联机作业和脱机作业。联机作业是指作业的完成是在用户的干预和控制下完成的，用户通过键盘（或图符）命令，一步一步地按作业要求的步骤，键入相应的命令完成相应的工作，直至最后完成作业的运行。例如，首先启动软盘，输入一个源程序文件，然后运行编译程序，将源程序翻译成目标代码，再运行连接装配程序，使其变成可执行代码，最后运行之。这时，用户直接控制计算机的运行，等待每一步运行的结束，再启动下一步的工作。人机交互频繁。这类系统比较强调系统的友好性。

脱机作业一般是指批处理作业，用户将作业（包括程序、运行数据及控制文件）交给操作员，由操作员将其输入计算机系统，然后在系统的控制下完成作业的运行。除了作业的输入需操作员干预外，其他的一切都是自动完成的。这时，需要用户向系统提供分阶段运行作业的步骤，甚至包括作业运行中出现异常情况时的处理策略，完全割裂了用户与计算机的交互。这种方式强调的是提高系统运行的效率以及计算机系统资源的利用率，通常用系统吞吐率（单位时间内系统完成作业的数量）作为指标。同时，批处理系统也用作业的周转时间来考核，所谓作业的周转时间是指作业进入系统到运行完毕的时间。现在，有时也把周转时间混用为响应时间。

联机作业就是通常在 PC 机上使用计算机的方式，这种方式强调的是用户的参与，强调用户友善性。这里，用户完全控制计算机的运行过程，机器运行的每一步骤都由用户通过命令（包括文字命令和图符命令）来加以确定。在计算机执行用户指令的过程中，用户可以随时终止此次指令的执行，并转发其他指令。因此，联机作业并不强调系统资源的高效利用，但为了用户使用感到方便，需要注意限制系统的响应时间，不能过长。所谓响应时间是指命令的响应时间，即一个命令从发出开始、指导命令执行完毕并返回结果的时间。

### 2. 作业调度

作业调度通常只指批处理作业。这类作业进入系统后并不立即运行，而是先作为后备作业存放在磁盘上，系统按照某种策略，挑选后备作业，将其调入内存并运行，谓之作业调度。作业调度必须具有在用户规定的作业完成最后时限以前完成作业的能力。

作业调度的出发点要兼顾几个方面。首先，它应该考虑尽量提高批处理系统的吞吐率，亦即尽量降低作业的平均周转时间；其次，它必须考虑避免由于片面追求吞吐率，而使调度算法失去在用户规定的作业完成最后时限以前完成作业的能力，亦即会引起“无限等待”现象。所谓无限等待，是指由于调度算法的某些缺陷，在一些特殊的情况下，



有可能使某一个或某一类作业在一个不可预计的时间内得不到运行。所谓特殊情况，往往是指当时系统运行的外部情况，对批处理系统而言，指的就是作业到来的情况。不过，“无限等待”与死锁不同，死锁指的是整个系统的行为，是结构性的，具有整体效果，它影响的是整个系统。死锁的产生当然是由于某个资源请求系列执行的结果，一旦发生死锁，这个结果就是不可逆转的，是不可能靠系统本身力量来解脱的。也就是说，死锁一旦形成，只能靠外力来消除。而“无限等待”指的是系统算法对单个或某一类个体潜在的不公平因素，是局部性的。当外部条件（这里就是到来的作业序列）的某些特性诱发了这些不公平因素产生作用时，会使某些用户处于长时间不能运行，而且无法估计未来的情形。但是，“无限等待”的情形是可以逆转的，一旦诱发条件消失，无限等待就不复存在。因此，随着时间的推移，“无限等待”现象可以自动消除。例如，某个排队算法允许有条件“夹塞”，虽然可以降低整个队列排队元素的平均等待时间（这对整个队列是有利的），但由于有可能符合条件的排队元素不断拥来，就可能使得某些排队元素在相当长的时间内无法得到服务，由于无法估计新到来排队元素的类型，它们何时能够得到服务也不能预计，形成“无限等待”现象。但是一旦新到来的排队元素并不符合“夹塞”条件，“无限等待”现象就会自动消失。

### 3. 用户界面

用户界面指用户使用计算机系统的方式，应该说目前一般意义下的用户界面主要是指联机方式。目前通用的是图形界面。操作系统界面是指用户要求操作系统提供服务的途径与方式。它一方面是指联机方式下的工作方式，另一方面也指在脱机方式下的工作方式，还指用户的程序运行时，需要操作系统提供服务的方式。因此，完整地理解操作系统的用户界面应该包括命令、系统调用和作业控制语言三个大类。

命令是操作系统向联机用户提供的操纵与要求操作系统提供服务的方式，通常是命令行加回车（Enter 键）。Enter 键的作用是产生一个中断，进入操作系统程序运行。通常，一个命令对应操作系统中的一个服务例程，操作系统就将命令理解为要执行相应的服务例程。目前流行的图形界面工作原理是一样的，单击某个图符，松开鼠标就发出一个中断。不同的是，命令可以有参数，但图符通常没有参数，图符的位置表示一个操作动作。当然，有时候参数也可以用随后的图符来代表。有些图符、参数则用涂黑了的内容来代替。所以实际上一条命令会对应一串图符单击动作。例如，打开文件只用一条命令：open 文件名。而用图形界面，先单击“打开”的图符，然后是一系列的文件目录单击动作以打开目录，最后单击文件名，完成打开文件。

接收键盘中断后操作系统开始运行，首先响应中断，分析是键盘中断，从键盘缓冲区读入命令内容，然后根据命令名，以命令参数作为调用参数，调用相应例程，完成用户要求的服。然后，再等待下一条命令的输入。

系统调用的形式与命令极为相似：调用名（参数），而且连起名亦大多相同。不过系统调用是潜入用户的程序中执行的。编译程序将系统调用语句翻译成一条程序性中断



指令（指令中可以附加调用号），并在紧接着该指令的存储单元放置参数（可能有多）。程序运行时，发出中断，操作系统开始运行，首先响应中断，分析是程序性中断，取出调用号，再从随后的区域取来参数，然后根据调用号，连带调用参数调用相应例程，完成程序要求的服务。相同名字的系统调用和命令调用的是同样的例程，差别仅在于一个是程序驱动，一个是键盘驱动而已。而图符则将执行过程分解得更细，用户操作虽然直观，但效率低。

作业控制语言（JCL）是操作系统提供的另一种要求其完成服务的界面，专门用于批处理作业，用它编写作业控制说明（就是作业控制程序），由作业控制进程解释执行。在联机运行时，可以将用作业控制语言编写的程序存入一个文件，然后执行这个文件，由用户控制进程解释执行。简单的作业控制程序就是一个命令串，顺序执行。复杂的作业控制语言（如 UNIX 的 SHELL 语言）具有与编程语言几乎相同的语句结构，但同样是解释执行的。作业控制程序组合多个程序协同执行，完成复杂的任务。

## 2.2 例题分析

### 2.2.1 操作系统内核与处理机管理

**【例 2-1】** 什么是操作系统的内核，简述内核在操作系统中的位置、作用和运行状态。

**解：**操作系统的内核直接构建在硬件之上，是硬件机器的第一级扩充。它主要涉及的硬件是 CPU，通过内核的管理，将一个物理的 CPU 转化为多个虚 CPU，即进程虚拟机。这样，内核就向上提供了一个支持多个进程运行，不再关心中断的运行环境。当然，内核必须负责中断的响应，负责各类寄存器内容的保留与恢复，调度 CPU 执行进程的程序，提供保证进程之间通信完整性的机制。此外，它还管理时钟，时钟是一类特殊的寄存器，它按固定的频率加 1 或者减 1，因此可以用来计时。在加 1 的情形，寄存器溢出时发中断，减 1 的情形则到 0 时发中断。前者叫正向时钟，后者称为反向时钟，两者的用途是等价的。

**【例 2-2】** 什么是计算机的状态？什么是特权指令？

**解：**计算机运行时的状态可以分为系统态（国内又称管态）和用户态（国内又称目态）两种。当计算机处于系统态运行时，它可以执行特权指令；而在处于用户态运行时，则不能执行特权指令，如果此时程序中出现特权指令，机器将会发出特权指令使用错误的中断。所谓特权指令集是计算机指令集的一个子集，特权指令通常与系统资源的操纵和控制有关，例如访外指令，用于启动通道；时钟控制指令用于取、置时钟寄存器的值；程序状态字控制指令用于取、置程序状态字；通道控制指令用于访问通道状态字；中断控制指令则用于访问中断字等。内核运行在系统态下，而一般来说进程则运行在用户态

下。内核程序的执行是由中断触发的。因此，进入操作系统运行首先进入内核运行，操作系统程序的入口地址就是内核程序执行的起始地址。

**【例 2-3】 简述中断机制及其与操作系统的关系。**

解：中断是操作系统实现多个程序并行的基础，在系统中处于枢纽的地位，操作系统任何程序的运行切换都是由中断处理的结果，在这个意义上，也可以说中断才会引起运行程序的切换。同样，任何运行中的错误，计算机都通过发出终端来报告错误，提醒处理。完全可以说，没有中断，就没有现代意义下的操作系统。

所谓中断，是指一个程序在运行过程中可以被打断，转去执行另一个程序的现象。有了操作系统，被打断的程序还可以在将来的某个时刻又在原断点恢复运行。中断本身是一个硬件概念，在每个指令执行的最后一拍，硬件将检查中断电平，如果发现是高电平，就表示发生了中断，这意味着有更紧急的事件需要处理。这时，硬件将响应中断，暂停原来执行的程序，交换程序状态字（PSW）。这样，机器就按新程序状态字指示的地址取指令并执行了。

围绕中断概念，有中断源，即产生中断的原因；中断字或中断向量，即中断源的集合。由于检测到中断发生时，可能有不止一个中断，因此，系统用一个字的形式，每一位表示一个中断源，一个中断到来的时候，相应位就置 1。这样就可以通过检查中断字来发现本次中断响应有几个中断，都是什么原因了。同样，中断处理后，相应位应该清零。

需要说明的是，中断是可以屏蔽的，当中断屏蔽时，中断电平不反映出高电平，但中断被保留，这可以通过门电路来实现。一旦撤销屏蔽，中断电平显示为高电平，再进行响应。这就是为什么可能一次响应多个中断的情形会出现的原因。

**【例 2-4】 何谓程序状态字（PSW），何谓交换程序状态字？**

解：所谓程序状态字，是指 CPU 一些重要寄存器内容的有序集合，而不是指硬件寄存器本身。计算机将这些内容排列起来，组成机器字的形式。PSW 包含两大方面的内容：第一部分是一些对计算机运行起重要控制作用的寄存器，如状态寄存器、中断屏蔽寄存器，各类控制寄存器等；第二部分是指令地址寄存器，指示将要执行下一条指令存放的内存地址。在内存中，计算机还特别指定两个大小与 PSW 一样的区域，一个由操作系统事先放入代表操作系统内核程序运行时应该要求的机器状态内容和内核程序起始地址，俗称新 PSW，另一个为空，俗称老 PSW。当硬件响应中断时，机器自动将这些硬件寄存器的内容放入后者，并将前者的内容填入相应的硬件寄存器，称为交换 PSW。然后，计算机将继续运行，从指令地址寄存器中取出内容，按其地址提取指令执行。显然，这时计算机执行的将不是原来的程序，而是操作系统的程序了。

**【例 2-5】 什么是多级中断，多级中断的计算机操作系统为什么设计了多个新程序状态字？**

解：有的计算机系统配备了多级中断的机制，即按轻重缓急程度，将所有的中断

源分为几个大类。设置了某一级的中断屏蔽，可以屏蔽本级及其以下级的中断，但仍可以响应更高级的中断。例如设备结束中断不太紧急，可以放在最低级，而断电等则必须立即处理，放在最高级。相应地，这时存放新 PSW 的区域就有多个，与中断的级数一致，里面存放的是响应该级中断时机器应具有的状态，而指令地址寄存器的内容都是一样的。

需要指出的是，新 PSW 的指令地址寄存器内容指示的就是操作系统程序的入口地址，这样一旦发生中断，计算机将转而运行操作系统的程序。而被中断的程序恢复运行时，操作系统将老 PSW 的内容填入相应硬件寄存器，就可以保证该程序从原断点恢复运行。还需要指出的是，中断发生后转入操作系统内核程序的运行，这时操作系统应该首先保留更多的内容，它应该保留老 PSW（否则在中断处理时再发生中断响应，就会破坏原来的内容），应该保留运算寄存器的内容，应该保留地址映射用的地址寄存器内容等。所有这些统称为运行现场，在被中断程序运行前一并恢复。

一般而言，中断处理期间需要屏蔽本级中断，这意味着还可以响应更高级的中断。显然，处理不同级别中断时计算机的状态（主要是屏蔽中断的触发器的状态）是不同的，这也就是为什么配备多级中断的计算机操作系统需要有多个存放新 PSW 区域的道理。

#### 【例 2-6】什么是中断处理和中断响应？

解：整个中断处理过程包括中断响应、中断处理和中断恢复三个部分。如果计算机没有屏蔽中断，一旦中断发生，计算机将自动响应。中断响应由硬、软件共同完成，硬件完成交换程序状态字的工作，软件则完成保留运行现场、分析中断原因等工作，明确了中断原因以后，就可以根据系统规定的策略，针对不同的中断原因进行不同的处理。

图 2-1 为中断处理全过程示意。

中断处理则完全由操作系统完成，分情况执行不同的中断处理例程，这些例程既可以在内核中调用执行，也可以调度进程执行。中断处理完成以后，有可能恢复被打断的程序运行，这时操作系统软件就负责恢复运行现场，被中断程序将从原断点恢复运行。

这里需要强调说明的是，操作系统程序只能通过中断进入并开始运行，这是激活操作系统程序的惟一手段。因为进程运行与内核运行时计算机分别处于用户态和系统态，它们之间的运行切换不能通过程序转移来完成（转移前后计算机的状态不会改变）。一旦发生中断并响应，一定首先进入操作系统的内核程序运行，然后根据需要，调度其他进程的运行。

还需要强调说明的是，中断处理可以调用进程执行，其中也包括调度本进程（发生中断时正在运行的进程）执行。发生这种情况往往是进程在运行时出现了由于本身执行

语义引发的例外情况。因此，进程程序也可以有多个入口。

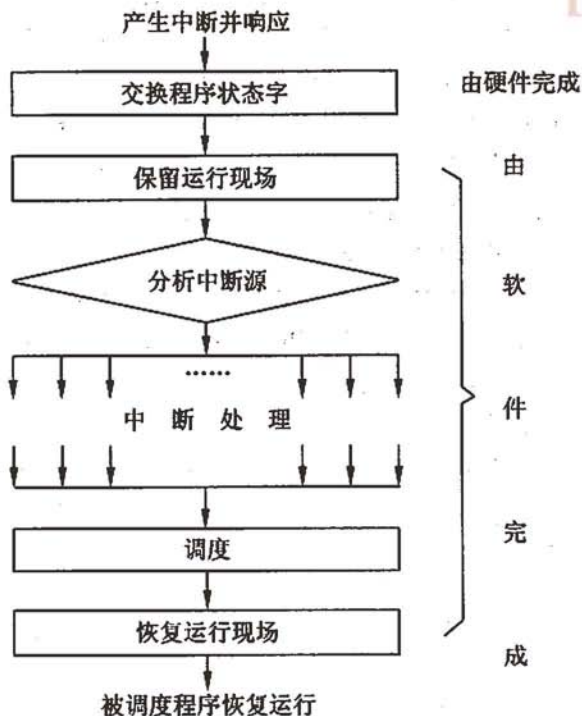


图 2-1 中断处理全过程

**【例 2-7】** 什么是程序？系统中只有一个程序时，系统运行有什么特点？

解：程序是指令或语句的有序集合，它们体现一个相对完整的运行逻辑或算法。

在冯·诺依曼型计算机里，指令是等同于数据存放在主存储器中，计算机的指令地址寄存器指示将要执行的那条指令的内存地址。在一般的情况下，计算机每执行一条指令，指令地址寄存器就自动加一，顺序指向下一条指令，保证了指令执行的逐条和顺序。即使是在出现转移（执行转移指令）的情况下，指令地址寄存器的值发生非顺序的改变，但这只是执行转移指令的结果，指令地址寄存器仍然指向即将执行的那条指令的内存地址，而且这种转移是在程序内部安排（指令地址寄存器的新值由转移指令给出）的，完全具有可预见性。显然，程序的顺序执行特性是由计算机硬件的特性（具体的是指令地址寄存器的功能）来保证的。在操作系统中，顺序的概念不局限于此，这里指的是 CPU 只执行一个程序的情形，这时，程序的执行不受任何外来的影响（故障除外）。

需要说明的是，主程序与子程序同属一个程序，因为它们之间具有从属关系，其运行切换点是事先设计固定的，只要把子程序“嵌入”主程序，它们就变成了一个



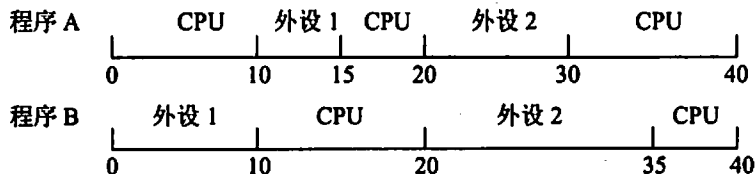
程序。

**【例 2-8】** 什么是多道程序环境，为什么要引入多道程序运行？

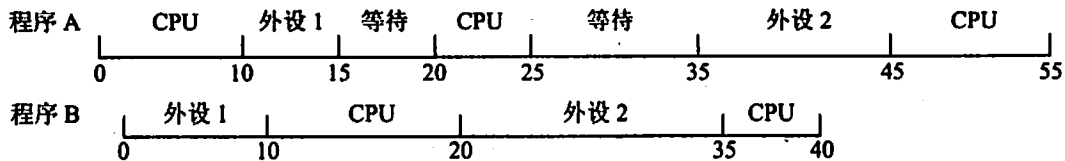
解：所谓多道程序环境就是并行环境，是指系统中同时有两个或两个以上的程序，它们的执行时间是重叠的。也就是说，其中某个程序可以在另一个程序运行没有结束时就开始运行，这些程序交替地穿插着使用 CPU 来执行自身程序。由于只有一个 CPU，在任何一个瞬间，系统中最多只有一个程序在运行。但是不排除有的程序在启动了外部设备以后，由于需要等待数据传输结束而不能继续运行，这时可以让别的程序运行，从而形成外部设备与 CPU、外部设备与外部设备之间的并行工作。

现代计算机系统的性能十分优越，资源相当丰富，一个程序的执行往往不能充分利用计算机系统的全部资源。例如，高速的 CPU 和速度相对较慢的外部设备之间，处理速度上的差距在不断地扩大。在只有一个程序运行的情况下，常常会出现计算工作需要等待输入/输出任务结束才能继续的情形，从而造成宝贵的 CPU 资源浪费。同样，计算机内存配置的不断扩大也使得单个程序难以全部使用系统的内存资源，外部设备资源也有可能因等待而造成浪费。而多个程序的并行就可以缓解这类资源浪费的现象，达到充分利用系统资源、提高运行效率的目的。

例如，有两个程序：



如果在顺序情况下，程序 A 执行结束后再执行程序 B，完成两个程序共需时间为  $40+40=80$  s，CPU 的利用率为 50%，外部设备 1 的利用率为 19%，外部设备 2 的利用率为 31%。如果在并行环境下两个程序同时运行，适当调整 A、B 的运行次序，只有在资源要求冲突时适当等待：



完成两个程序共需时间为 55 s。这时 CPU 的利用率提高为 73%，外设 1 的利用率提高为 27%，外设 2 的利用率提高为 45%，显然资源的利用率得到了很大的提高。

由此我们可以看出，引入多道程序机制以后，可以实现计算和输入/输出的并行（上

例中的前 10 s)、外部设备之间的并行（上例中反映不明显），从而既提高了资源的利用率，又提高了系统的吞吐率。

**【例 2-9】** 什么是互斥？如何理解正确实现互斥的关键问题？

解：互斥是一种最简单的制约关系，主要在并行环境中多个程序对共享数据区进行操作时出现。显然，对某个共享数据结构进行操作的程序代码段是散布在各个程序中的，我们把涉及同一个共享数据结构的所有程序片段的集合称为“互斥区”。图 2-2 的阴影部分即为互斥区。

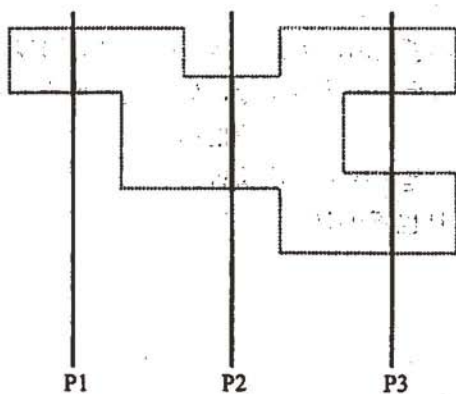
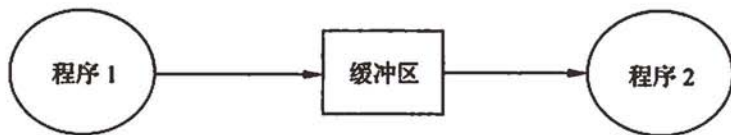


图 2-2 互斥区示意图

我们把满足下述三个条件的互斥区称为“临界区”，也就是说，临界区是满足“有空让进、无空等待、有限等待”原则的互斥区。它的实际意义是显然的。例如有一个数据结构，程序 1 写入数据，程序 2 去读出来：



如果程序 1 和程序 2 的读、写操作不加互斥，在写到一半的时候就去读，只能读出一半，将不能读出完整的数据。

用常规的程序设计方法解决临界区问题是可以实现的，但非常繁琐，而且需要“忙等”，影响效率。我们在下面将引入“信号量”机制，就可以方便地解决这个问题。

**【例 2-10】** 什么是同步？如何理解正确实现同步的关键问题？

解：并行环境中的几个程序必须协调自己的运行速度，以保证各自的某些关键语句按照某种事先规定的次序依次执行，这种现象称为同步。仍然看上面的例子，假定程序 1 不断地写缓冲区，程序 2 不断地读缓冲区，以如下程序表示：

## 程序 1

```
loop {  
    产生数据;  
    写入缓冲区;  
}
```

## 程序 2

```
loop{  
    读缓冲区;  
    处理读出的数据;  
}
```

这里的关键语句是写入缓冲区和读出缓冲区，它们需要同步。正常的次序应该是写入以后才能读（这样才能读出完整的数据）和读出以后才能写（这样才能保证不会丢失数据）。对执行次序的规定称为同步条件。本例中有两个同步条件。

显然，同步中已经蕴含了互斥，因此互斥是最简单的同步关系。这时，只规定某些语句必须在一个执行完成后另一个才可执行，但不规定特定的执行次序。

**【例 2-11】 什么是进程？如何理解进程的概念？**

解：进程的定义：进程是程序的一次执行。

这个定义太抽象，而且也不贴切。我们需要从进程的涵义与外延两个方面来进行讨论，不仅要讨论进程的行为，还必须讨论进程的支持机制，这样才能完整地理解进程。

进程的行为与上面所述的处于并行环境中的程序的行为类同，它们可以在运行中停顿，还可以在断点恢复运行，进程之间存在复杂的同步关系，实现彼此之间的协调等。

首先考虑进程的涵义。进程包括程序和数据是显然的，程序用来反映进程需要完成所规定任务的实现逻辑，通过程序的执行来完成任务，当然是进程运行的主体。数据是程序运行是必需的，自然也无疑问。此外，还有一个重要的部分是进程控制块（PCB）。进程控制块是内核中最重要的数据结构，它是进程存在的惟一标志，它描述了进程的基本情况，PCB 记录的进程基本信息可以分为两大部分：调度信息和运行信息（又称现场）。调度信息根据不同的计算机和操作系统而不同，但必须包括进程的标识，例如进程名（或进程号），需要的话还可以有用户名、组名等，进程的状态、进程的优先数、进程的队列指针（进程排队时使用），以及一些统计与调度信息，例如进程占用 CPU 的时间、未占用 CPU 的时间等。这些信息将在调度时使用。运行现场则在进程暂停运行时用来保留断点的现场。原则上，运行现场应该包括进程的各类数据、打开文件，以及计算机的各种寄存器内容等。但是，由于进程有专用的内存区域，其他进程运行时不能访问并破坏进程的专用存储区，因此不必另行保留，PCB 保留的只是其他进程运行时会遭到破坏的那部分内容，包括 PSW、各类运算寄存器、地址寄存器等。为有多重中断的计算机配置的操作系统，现场也有多重。所有的 PCB 放在核心的 PCB 区内，组成一张 PCB 表。通过队列指针将 PCB 连接形成许多队列。因此，PCB 区的大小也就决定了可以创建进程的最大数目。表 2-1 表示一个简化的 PCB 区。

表 2-1 简化的 PCB 区

进 程 名	优 先 数	状 态	队 列 指 针	统 计 信 息	运 行 现 场
A	40	Wait	E	Any	Any
B	37	Run	Nil	Any	Any
C	60	Wait	G	Any	Any
D	54	Ready	Nil	Any	Any
E	65	Wait	Nil	Any	Any
F	70	Ready	D	Any	Any
G	66	wait	Nil	Any	Any

从表中看出, 这里共有 7 个进程, 形成了 4 个队列, 运行队列中只有一个进程 (单 CPU 的计算机最多如此); 就绪队列有一个, 其中有两个进程, 排队次序为 F、D; 等待队列有两个, 其中一个队列中的排队次序为 A、E; 另一的队列的排队次序为 C、G。

其次, 再来看进程的外延。进程的外延指的是为保证进程行为的正确实现, 操作系统内核所必须具备的支持机制, 包括处理机调度、现场管理、队列管理、中断处理、通信管理等。几乎可以说, 内核的所有程序都将围绕进程的正确实现及其运行而提供服务支持。其中, 处理机管理将保证 CPU 在进程之间切换, 实现进程的并行运行; 现场管理将保证进程的运行可以随时停顿, 并将在以后从原断点正确地恢复运行; 队列管理将保证进程在等待、就绪和运行之间正确地转换状态; 中断处理将保证在进程运行时, 系统可以随时正确地处理中断; 通信管理将保证在进程之间正确地实现同部、互斥关系, 实现正确的制约等。这样, 就呈现出在内和的管理、协调下, 多个进程此起彼伏地并行工作的系统行为。

#### 【例 2-12】 进程如何创建?

解: 进程的创建主要通过以下两种途径。

首先是在操作系统初始化的时候创建进程。操作系统初始化包括载入操作系统的程序和运行操作系统的初始化程序, 这时, 将生成若干个进程, 并将它们投入运行。

其次是在进程运行时, 可以生成子进程, 子进程运行时还可以生成子进程, 从而形成进程家族, 可以用一棵树来表示。操作系统中提供生成子进程的系统调用, 例如 UNIX 的 fork。子进程可以继承父进程的资源, 但在进程管理上, 它们是平等的。同样, 父进程可以终止子进程并将其撤销, 操作系统同样提供这样的系统调用, 例如 UNIX 中的 kill。

创建进程的主要工作是填写进程控制块, 然后将它们链入就绪队列, 以便投入运行。

#### 【例 2-13】 什么是进程的状态, 进程的状态之间如何切换?

解: 进程的状态可以分为两大类: 活跃状态和非活跃状态。进程在创建和撤销过程中所处的状态称为非活跃状态, 这时系统正在填写它的 PCB 或者正在回收其资源, 其结局或者是进程创建完成, 进入活跃状态, 或者进程被撤销 不复存在。创建完成和执行撤



销操作之前，进程处于活跃状态。我们关心的是处于活跃期内的进程行为。

这时，进程的状态可以有三种：运行态、就绪态和等待态。进程占有 CPU，正在运行其程序时处于运行态；进程具备了一切运行的条件，但由于 CPU 正在运行别的进程而使它不能运行时，处于就绪态；进程由于自身的原因必须等待某个条件的具备，否则不能继续运行时，处于等待态。显然，就绪态实质上是一种特殊的等待态，即等待 CPU 资源的等待态。进程在活跃期内，不断在这三种状态之间切换，于是在外部行为上就呈现出走走停停的形态，处于运行态时，进程向前行进，而处于等待态和就绪态时，进程处于停顿。

进程状态之间的切换如图 2-3 所示。

运行进程由于某个条件不能满足，例如申请的打印机正在为其他进程所用而不能使用，这时不具备继续运行的条件，必须等待对方打印完毕，自己得到打印机资源后才能恢复运行，这时由运行态变为等待态；一旦等待条件消失，例如进程

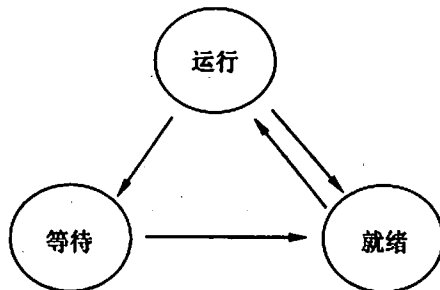


图 2-3 进程状态切换图

获得了资源，具备了继续运行的条件，就由等待态变为就绪态；而就绪态的进程一旦被调度获得 CPU，自然就变成了运行态。还有一类特殊的情况，运行进程虽然具备继续运行的条件，但自愿放弃或被强制剥夺 CPU 而不再继续运行，这时由运行态变为就绪态。例如，进程为了让某个特定的进程先运行，可能会放弃 CPU，在分时系统中，运行进程分配的时间片到，这时就会被强制剥夺 CPU，供其他进程运行。

**【例 2-14】** 操作系统中的进程队列有哪几种？队列管理涉及哪些内容？

解：所谓队列，就是进程的队列，队列的元素就是进程。进程排队实际上是 PCB 在排队，通过 PCB 的队列指针，若干个进程控制块组成一张链表。队列也是核内核的数据结构，它只是一个指向队首元素的指针。操作系统的队列分为三类：运行队列、就绪队列和等待队列。在单 CPU 情形，运行队列只有一个，而且退化为最多只有一个元素，也可能为空。一般来说，就绪队列也只有一个，有的系统也设两个，例如 UNIX，一个是真正意义上的就绪队列，另一是指那些等待原因已经消除，但却被对换至外存的进程队列，实际上，这是一个特殊的等待队列，一个等待被对换到内存的进程队列。等待队列可以有多个，一个等待原因就是一个等待队列，例如，等待扩大内存的进程就处于同一个队列等。任何进程在任何时刻处且仅处于一个队列之中。

队列管理的操作有入列、出列、转列和队列整理四种。所谓入列是将一个 PCB 链入队列，可以是放在队尾、放在队首或插在中间（例如，按优先数插入队列）。所谓出列，就是取出队首元素。转列是将一个队列转入另一个队列，通常是将一个等待队列转入就绪队列，例如，当某个进程释放一片内存时，就会将等待分配内存的队列转入就绪队列。队列整理在使用动态优先数的系统中使用，系统重新计算优先数后，每个进程的优先数

都有可能改变,这时就要整理那些基于优先数排队的队列,以便按照新的优先数来确定排队次序。

一个在单CPU计算机上运行、具有 $n$ 个进程的操作系统,当处于非内核程序运行时,运行队列中最多具有1个进程、就绪队列中最多有 $n-1$ 个进程,等待队列中的进程数量则不受限制。由于系统只有一个CPU,因此最多只有一个进程在运行,也有可能此时所有的进程都处于等待状态(但不是死锁),例如等待数据传输结束,这时没有运行进程,因此其数量范围为 $0\sim 1$ 。就绪进程不可能有 $n$ 个,因为系统处于非内核程序运行,如果有就绪进程,其中的某一个一定已被调度投入运行,因此其数量范围为 $0\sim n-1$ 。等待进程的数量没有限制,因为各种情况都可能出现,或者没有等待进程(这时就绪进程 $n-1$ 个,运行进程1个),或者全部进程都在等待,因此其数量范围为 $0\sim n$ 。

**【例 2-15】** 如何管理进程的现场? 应该注意哪些问题?

解: 进程管理的内容之一是现场管理,现场管理操作包括保留现场和恢复现场。进程运行时一旦发生并响应中断,操作系统内核首先要保留进程运行的现场。当进程被调度再次运行前,核心将为其恢复现场,这样才能保证进程从原断点开始恢复运行。这里主要应该注意严格保证保留、恢复的次序。保留现场的原则是必须真正的现场。由于保留现场也是由操作系统程序来完成的,操作系统内核程序本身的运行也会使用一些寄存器,因此保留的次序应该是易破坏的先保留,例如,运算器一定是第一个保留的。恢复现场也一样,必须保证已经恢复的现场不被破坏,这样易破坏的应该后保留。但是,PSW一定是最后恢复的,因为一旦重新置了PSW,机器下一条指令执行的是进程的程序,离开了内核运行。

**【例 2-16】** 进程之间如何通信? 什么是进程之间的通信机制?

解: 进程之间的通信实际上就是进程之间交换数据,即某个进程发送数据,另一个(或几个)进程接收数据。操作系统的进程通信机制关心的是数据交换的正确性,实际上就是保证实现同步、互斥关系的正确实现,并不关心数据的内容。进程之间的通信机制可以分为两大类:利用共享存储区的数据交换和消息传送。所谓利用共享存储区的数据交换,就是某个进程向数据区写入数据以后,一个(或几个)进程到同一个数据区将其内容读出。这样做的突出特点是效率高,速度快。但是,这里的一个先决条件就是这些进程对该存储区必须都有访问权。显然,这种机制只能适用于单计算机环境,在网络环境下并不适用。所谓消息传送,就是某个进程发送消息,一个(或几个)进程接收消息。这种机制效率较低,但适用于网络环境。

需要指出的是,现代操作系统都统一使用消息传送的方式。但是,在单机环境下,消息传送通过缓冲区方式实现。这样做的好处是既统一了通信的接口和方式,适应网络环境,又可以在单机环境下提高效率。

**【例 2-17】** 什么是信号量? 什么是P、V操作? 如何理解与使用?

解：信号量是核心内部定义的一类特殊的数据结构，具有如下形式：

```
struct semaphore{  
    int s;  
    struct pcb *pcb_pointer;  
};
```

其中，整型域分量  $s$  表示该信号量的值，指向进程控制块的指针域分量  $pcb\_pointer$  表示与该信号量相关的进程队列指针。需要强调指出的是，信号量变量是由操作系统核心来管理和维护的，在信号量上的操作只能在核心内部来实现。严格来说，对一个信号量的分量的访问应该使用受限表示法；但有时为了方便，在叙述时常常将对信号量整型分量的操作直接用信号量的变量名来表示（本书中也是如此）。

信号量上的操作只有两种：P 操作和 V 操作。它们都是在操作系统核心内实现的。因此，进程的程序中虽然出现诸如  $P(s)$ 、 $V(mutex)$  那样的调用语句，而真正的计算工作却在核心中完成，进程不能对信号量实施任何运算。此外，信号量初值就是指其整型分量的初值（以后在本书中不再区分），也只能在核心中赋予（可在操作系统初始化时进行）。

```
P(s):  
    s := s - 1;  
    if s < 0 then  
        调用进程进入相应队列等待。  
  
V(s):  
    s := s + 1;  
    if s <= 0 then  
        从相应队列中唤醒一个进程，调用进程进入就绪状态。
```

信号量机制的表达能力是足够的，通过信号量和 P、V 操作的组合，可以解决操作系统设计中遇到的任何复杂的同步问题。实际上，信号量的整型分量是有实际意义的， $s$  的初值代表一开始可以允许多少个进程同时进入对共享数据结构的操作，经过一定时间运行后，若  $s$  的值为正，代表可以还有多少个进程可以进入对共享数据结构的操作，若  $s$  的值为负，表示有  $|s|$  个进程在相应的队列上等待。

在利用信号量机制设计同步问题解决方案时，一般来说，一个信号量可以解决一个同步条件，因此，一个同步问题中有几个条件，就原则上需要设立几个信号量。

需要强调说明的是，同一个信号量上的操作必须互斥执行，否则就会产生错误。所以，P、V 操作有时也叫做原子操作。意味着在这种操作执行中不能打断而去为另一个进程执行同一个信号量上的操作。还需要说明的是，在实现信号量机制时，可以采用全局互斥或局部互斥的方法。前者就是使得所有信号量上的 P、V 操作都互斥（这样当然



保证了同一个信号量上 P、V 操作的互斥), 实现也比较简单, 只要在信号量操作时屏蔽中断即可; 后者需要引入特殊的硬件指令, 如取后加 1, 取后减 1 等。

在进行设计时, 要注意同一个信号量上的 P、V 操作一般来说应该配对。此外, P、V 操作使用不当有可能造成死锁。例如,

$s_1$ 、 $s_2$ 的初值均为 1, 两个进程:	进程 1	进程 2
	⋮	⋮
	P( $s_1$ )	P( $s_2$ )
	⋮	⋮
	P( $s_2$ )	P( $s_1$ )
	⋮	⋮
	V( $s_2$ )	V( $s_1$ )
	V( $s_1$ )	V( $s_2$ )

这时, 就有可能死锁。因为, 若两个进程先后都在对方尚未执行第二个 P 操作以前执行了第一个 P 操作, 那么大家都无法通过第二个 P 操作, 都在等待对方执行相应的 V 操作唤醒自己, 形成死锁。

信号量机制的描述能力是充分的, 可以解决操作系统中所有的同步问题, 是一种非常重要的通信支持机制。

**【例 2-18】** 什么是进程调度? 进程调度的算法分为几类?

**解:** 进程调度又叫低级调度。进程调度的任务就是取出就绪队列的第一个元素, 将其投入运行。进程调度程序一般在中断处理结束后运行。因为一次进程调度的有效时间范围是两次中断之间的时间间隔, 所以又叫短程调度。进程调度的策略可以分为两大类: 抢占(剥夺)式调度和非抢占式调度。非抢占式调度是指一个进程一旦被调度, 除非由于自身的原因不能运行, 就将一直运行下去。也就是说, 如果当前运行进程的状态不改变为等待, 进程调度就不进行。而抢占式调度则按照某种评价准则, 在每次中断处理结束后进行调度, 选择条件最优的进程投入运行。亦即一旦发生中断并进行处理后, 如果当前进程未插入等待队列, 则将其按照优先数插入就绪队列, 然后将就绪队列中的第一个进程投入运行。所以, 在抢占式调度情形, 每次中断处理结束后都要运行进程调度程序, 而在非抢占式情形, 只有在中断处理引起当前运行进程状态改变时才运行调度程序。

进程调度常用的策略是优先数调度。这里又分静态优先数和动态优先数两种, 静态优先数是指进程的优先数在运行完全确定, 运行过程中保持不变, 而动态优先数则表示在进程运行过程中优先数是经常改变的。例如, 一个进程占用 CPU 的时间较长, 其优先级别将适当降低, 相反, 进程等待/就绪时间较长则适当提高优先级别, 从而保证某种公平性。一般来说, 采用动态优先数的系统每一个进程都有一个基准优先数, 运行过程中



以此为基础上下浮动。系统根据规定的算法，定期计算优先数，然后整理就绪队列。因此，进程调度的策略实际上体现在就绪队列的管理上。在这个意义上，进程调度程序包括三个部分：按某种算法计算优先数，整理就绪队列，将就绪队列的头元素投入运行。

**【例 2-19】** 什么是原语管理？如何实现？

解：在习惯上，把需要互斥执行的 P、V 操作称为原语。由于信号量是内核的数据结构，因此 P、V 操作将由内核程序进行。以 P 操作为例，虽然 P(s) 出现在进程的源程序中，但是经过编译后，P(s) 变成如下指令串：

```
trap x
s
```

这里 trap 表示捕获指令，其执行结果时发出程序性中断，该指令可以附带一个号码（一般为整数），此号码的语义由操作系统设计者约定（例如将代表 P 操作的号码定为 x）。下一条指令的位置放置参数，这里是信号量的名字 x。程序执行到 trap 指令时，计算机产生中断，转入内核程序运行，发现这是程序性中断，其附带号码为 x，代表是 P 操作，内核程序再取出参数，知道是 P(s)，于是就调用 P(s) 的例程，将 s 取出，进行减 1、判断等如上面已经介绍的一连串运算。由于原语操作非常简短，一般就在内核中进行，P 操作和 V 操作的操作例程称为原语管理程序。

必须指出的是，在用户程序中使用操作系统的系统调用时，系统调用的执行与原语执行的过程是一样的，不同的只是捕获指令所附带的号码不同，参数不同，系统调用的执行不一定都在内核中完成而已。以一条打印数据的系统调用为例，write(x, p) 表示在设备 p 上打印数据 x，经过编译后的指令串为

```
trap t
x
p
```

前面的过程与原语操作一样，根据附加号码知道是打印操作，取出参数知道打印设备是 p，打印的数据是 x，内核程序就调度管理打印设备 p 的进程运行，完成打印动作。

**【例 2-20】** 什么是时钟寄存器？什么是虚时钟？

解：时钟是一类特殊的寄存器，它以固定的频率加 1 或者减 1，在加 1 的情形，当时钟寄存器溢出时将发生中断，在减 1 的情形则当时钟寄存器为零时发出中断。由于频率是固定的，所以可以用来计时。前者称正向时钟，后者称反向时钟。对时钟的操作有两种：取时钟寄存器的值，向时钟寄存器送值，它们都是特权指令。

需要说明的是，正向时钟与反向时钟在使用上的能力是等价的。例如，需要使用闹钟时，若用反向时钟，将时间 T 送入时钟寄存器，这样过了时间 T 后时钟寄存器将会发

出中断，达到闹钟的功效；如果使用正向时钟，只需将 T 的补值送入时钟寄存器就可达到同样的效果。

虚时钟实质上是一个存放时间信息的内存单元。例如进程 CPU 使用时间就是一个虚时钟，为每个进程指定一个整型数据结构（例如称为 T），初值为零。当调度该进程运行前，将 T 的值送入一个正向时钟寄存器，一旦发生中断，立即取出时钟寄存器的值，放回 T 中。这样，T 中的值就记录了该进程累计运行的时间。用这种方法，一个硬件时钟就可以转化为多个软件时钟。

时钟管理通常也在内核中实现。

**【例 2-21】** 什么是分时系统？如何实现时间片轮转？

解：在系统形态上，分时系统中一个主机连接若干个终端，如图 2-4 所示。



图 2-4 分时系统示意图

在操作系统结构上，系统设置了若干个用户进程，每个用户进程控制一个终端，运行时的默认输入来自终端，默认输出也发向终端，用户与主机通过终端进行交互。分时系统的特色在于对用户进程采用时间片轮转的调度算法。系统设定一个时间值，称为时间片，在系统初始化时可以调整。系统将用户进程编号，按编号顺序调度，每当调度一个用户进程运行时，将时间片值放入闹钟，闹钟响时（时间片到发出中断），将运行进程插入就绪队列尾部，调度第一个就绪进程运行，就绪队列按进程号排列。如果运行过程中进程需要启动外部设备进行数据传输，则将其放入相应的等待队列，设备结束后在改变它的进程状态，按进程号插入就绪队列，忽略时间片的残值。

**【例 2-22】** 简述进程与程序的异同。

解：程序由代码（语句或指令）组成，是一个静态的概念，可以反映计算机的执行序列。程序行为就是其代码序列被执行时所体现的算法行为（只受到输入数据的影响），是确定的。其运行依托为硬件计算机。而进程则是一个动态的概念，进程的执行主体是它的程序，其运行依托不仅包括硬件计算机，还包括操作系统的内核。因此，程序行为只是进程行为中的功能部分（当然应该是确定的），其运行部分将受到系统中其他进程的影响，是不确定的。实际上，进程的动态性就体现在它的运行环境上。环境的变化虽然不应该影响其功能行为，但将直接影响其运行行为。在这个意义上，进程必须与操作系统的内核共存，因为它只能在内核的支持下才能运行并正常地完成其功能。“进程是程序的一次执行”作为定义失之于过分简略。

### 2.2.2 存储管理

操作系统中的存储管理主要包括内存管理和虚拟存储两大内容，外存管理通常放在文件系统中讲授。

由于用户程序是通过操作系统进入内存并由操作系统加以管理，程序的编制者和编译程序根本无法预知操作系统会将程序及其运行数据存放在什么地方，更何况目标代码需要经过连接装配才变为可执行代码。编译程序只能从零开始安排数据区和程序区，直到操作系统为其分配存放区域后才将地址映射到实存地址。因此，用户程序（包括经过编译后的目标程序片断）编址都是从零开始（称为相对地址），而在连接装配过程中（甚至在程序执行过程中）转为反映其实际存储位置的地址（称为绝对地址），为计算机硬件所识别、所访问和运行。将用户程序中出现的相对地址（又称逻辑地址）转化为计算机可以直接寻址的绝对地址（又称物理地址）的过程称为地址重定位。程序运行前完成全部重定位过程称为静态重定位，在程序运行过程中就出现的操作数逐个进行重定位称为动态重定位。动态重定位工作是由软硬件共同实现的，操作系统准备与提供定位依据（数据），硬件完成重定位动作。

内存管理是描述相对地址的形式以及重定位策略，包括所需要的硬件设施和软件策略。内存管理的策略大体上分为静态和动态两种，静态策略比较简单，将内存分为若干个长度固定、用途固定的分区，按内容分别存放。现代操作系统基本上都实现动态策略。动态策略分为段式管理、页式管理和段页式管理三种。段式管理采用内存动态分割、段内连续分配的方式，根据用户要求动态分配和释放内存；页式管理采用内存固定分割、根据用户要求以页为单位分配与释放内存；段页式管理则结合段式管理和页式管理的特点，采用内存固定分割为页、离散页组合成段、以段为单位申请与释放内存。

虚存管理是一种典型的由操作系统实现资源转化的例子。通过操作系统的虚存管理，CPU 资源和外存资源可以转化为内存资源，从而扩大系统的编址空间。利用程序的局部性特点，采用内外存交换技术，可以向用户提供比物理内存更大的虚存空间。

**【例 2-23】** 什么是固定分区技术？DOS 系统是如何实现固定分区的？

解：最简单的固定分区是单用户操作系统中的分区方法。例如，PC 上 DOS 将内存分为三个区：BIOS 区、用户区和系统区。BIOS 区位于内存的最高端，一般是只读的，用户区处于中段，系统区位于最下端。如图 2-5 所示。

多用户操作系统中的固定分区一般将系统区置于内存的一端，将其余内存划分为若干个大小不等的区，在

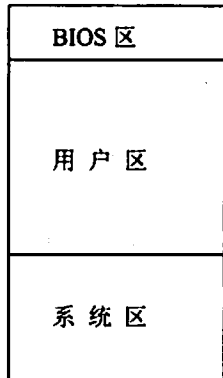


图 2-5 单用户操作系统内存分区

系统初始化时完成,系统运行期间保持不变。系统运行时,一个用户可以占用一个区,若无足够的用户,有的分区可以空闲。系统设立一张分区表指示各区的使用情况。当一个用户到来时,系统在分区表内为其寻找一个大小既能满足它要求又空闲的区加以分配,用户离开时,将分区表上对应的表项置为空闲。这种管理方法简单,但适用范围窄,静态方式难以满足不同用户的要求,如图 2-6 所示。



图 2-6 多用户操作系统内存分区示意图

固定分区系统的地址重定位可以采用静态方式,在用户进入时完成。

**【例 2-24】** 段式管理的策略是什么? 段式管理如何实现地址映射?

解: 段式管理采用内存动态分割技术,允许用户一次使用多段存储区用来放置程序和数据。用户申请内存以段为单位,当用户申请一段内存时,系统将根据用户的要求为其在内存中切割一段存储区进行分配。因此,一个段就是一个地址连续的内存区域。

采用段式管理的系统,用户程序里出现的相对地址为:

段 号	段 内 地 址
-----	---------

段式管理需要硬件的支持。计算机硬件必须配备界地址寄存器,界地址寄存器是成对出现的,一对界地址寄存器包括基址寄存器和限长寄存器(或上界寄存器和下界寄存器),基址寄存器存放该段的起始地址,限长寄存器存放该段的长度。硬件机器配备界地址寄存器的数目就是每个程序所能够使用段的最大数目。界地址寄存器组成一张表,称为硬件段表。同样地,进程应该设立软件段表,称为进程段表,进程段表放在 PCB 中作为现场的一部分。当进程运行中响应中断时,应把硬件页表的内容作为现场的一部分保留;同样,进程恢复运行前要将进程段表的内容恢复到硬件段表中。

程序执行过程中,由硬件实现从相对地址到绝对地址的映射。当一个相对地址到来时,取出段号,找到相应的界地址寄存器对,将段内地址与限长寄存器的内容比较,若



段内地址比限长寄存器的内容大，则表示地址越界，发“内存错”中断，否则将基址寄存器的内容与段内地址相加，得到绝对地址，完成地址映射。地址映射的流程如图 2-7 所示。

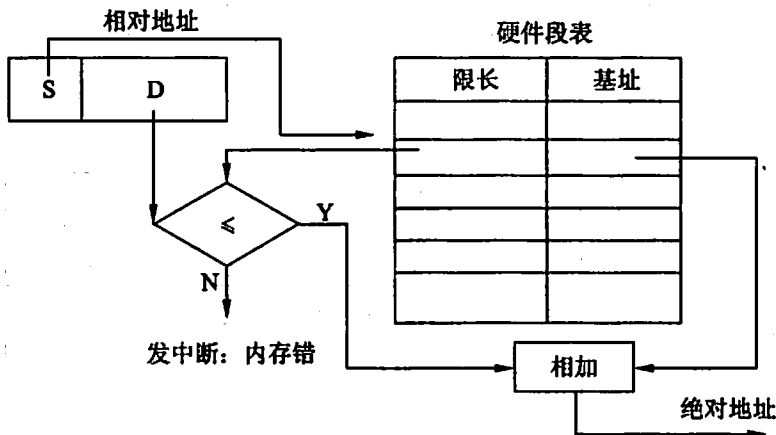


图 2-7 段式管理地址映射

因此，段式管理的特点是连续分配（段内连续）、动态切割（段边界随意），允许用户运行过程中增加并使用多段内存区域。这种方式比较符合一般程序的使用习惯和要求，因为大量的程序是分程序嵌套的结构，进入分程序将申请一段存储，退出分程序时释放它。

**【例 2-25】** 段式管理能否实现内存资源的充分利用？为什么？

**解：**段式管理并不能实现内存资源的充分利用，遇到的问题主要是“碎片”问题。经过长时间的运行，内存被反复地切割以响应用户不断提出的申请、释放（内存）要求，内存可能变得支离破碎，出现大量的互不邻接的空块，每个空块长度不大，基本不能满足用户的正常需要。然而，所有空块加在一起，又是一个相当大的数目，造成内存不能充分利用。由于碎片在段以外，这种碎片又称“外碎片”。

为了解决碎片问题可以通过系统紧致，即全面移动系统中的用户程序和数据，同时修改有关地址寄存器的内容，将正在使用的段向内存的一个方向集中，剩下的另一个方向就是连成一片的空闲区了。不过，系统紧致需要消耗大量的 CPU 时间，紧致时所有用户的活动都必须停止，包括输入输出在内。

**【例 2-26】** 简述页式管理策略及其地址映射策略。

**解：**页式管理是将内存固定分割为若干页，页的边界完全固定。例如，一个系统的内存为 4M，每页的大小为 4K，这样系统就由 1K 页，页的边界为 4K、8K……

在页式管理的系统中，相对地址的形式为：

页号	页内地址
----	------

绝对地址同样可以这种形式来理解,不过,绝对地址的页号就是物理页号,而相对地址中的页号是相对页号罢了。进程应该建立一张进程页表,它维护相对页号与绝对页号的对应关系。同样,硬件应该设立一张硬件页表,用以运行过程中的地址映射。根据相对页号找到硬件页表中的对应表项,然后取出绝对页号与相对地址中的页内地址拼装,即可得到绝对地址。见下图 2-8。

**【例 2-27】** 什么是进程页表?为什么要引入联想寄存器?

**解:** 记录一个进程所属全部页面的相对页号与绝对页号之间对应关系的数据结构称为进程页表。进程页表是一个相当大的数据结构,配置较小的计算机系统没有足够大的硬件页表放置整个进程页表,这时只能在硬件页表中放置一部分,地址映射时发现找不到相对页号时发出中断,由软件查找进程页表完成地址映射,同时,将选择硬件页表中的某个表项放回进程页表,用刚才的表项取而代之。这样,硬件页表的命中率对运行性能的影响就相当大。同段式管理一样,进程的硬件页表也是现场的一部分。

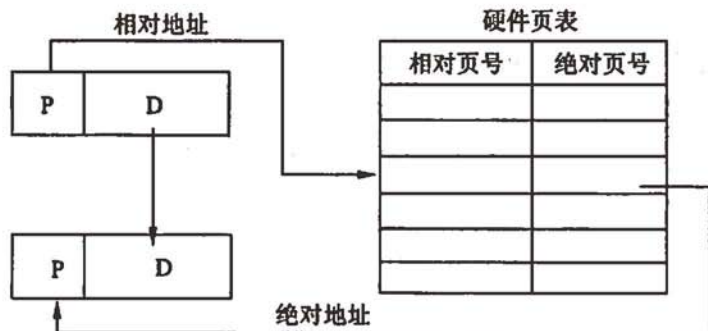


图 2-8 页式管理地址映射

现代计算机中,硬件页表由联想寄存器(又称联想存储器)组构成。联想寄存器的主要特点是并行查找能力,它无需软件进行逐个表项的比较来确定是否命中,通过一次访问即可。其次,它的访问速度也比内存访问快得多,还能实现联想寄存器和它相对应的内存地址同时修改。原来,联想寄存器相当昂贵,一般系统不便配置太多,现在 cache 技术的发展使得联想寄存器组可以配置的相当大,甚至包括全部软件页表。

**【例 2-28】** 页式管理时,如何管理内存?

**解:** 为了管理内存,操作系统应该设置一张表,记录每个物理页的使用情况,该表被称为“页示图”。当一个申请要求到来时,系统查找“页示图”,找出一个空闲页,将其分配,并修改“页示图”相应的表项为“使用”,并在进程页表中记录相对页号与绝对页号

的对应关系；当释放某一页时，只需根据进程页表，将“页示图”上相应的表项改为空闲、删除页号对应关系即可。通常，页示图应该放在内存。但是，由于内存可能配置很大，这样页示图会变得很长，这时，可以将它全部放在外存（通常是磁盘），在内存开辟一个区域，存放一部分页示图表项，并记录其页面的起始物理页号。申请要求到来时，寻找空闲页加以分配，若无空闲页，则将内存中的页示图存回磁盘，调入一块新的页示图，再行寻找。释放要求到来时，若待释放页的页示图表项位于内存，则删除页号对应关系并将相应表项改为空闲；否则，将内存中的页示图存回磁盘，调入相应的页示图，再行修改。

**【例 2-29】** 页式管理会不会产生“碎片”？为什么？

解：页式管理内存固定分割，管理比较简单。虽然它不会引起外碎片，但是，分配存储的基本单位是一个页，进入分程序时，即使局部变量不足一页，也必须申请一个页，这时会产生内碎片，页式管理的平均内碎片长度为  $1/2$  页，而且使用也比较麻烦，需要将一段内存需求（目前的编程语言多用分程序嵌套语言）转化为若干个页，然后再申请。所以，很少有实际操作系统使用页式管理，它往往是作为虚拟存储的支持机制出现。这时内外存交换单位长度固定，实现起来比较方便。

**【例 2-30】** 什么是段页式管理？如何实现地址映射？

解：段页式管理兼顾了段式管理和页式管理的优点，是目前计算机系统最常用的内存管理模式。在用户层面，申请与释放都是以段为单位的，但在内存分割和管理方面，又是以页为单位的。在实现上，段页是管理的相对地址与段式管理相同，即由段号与段内地址构成：

段号	段内地址
----	------

但是，系统将段内地址解释为页号和页内地址：

段号	页号	页内地址
----	----	------

在实际存储分配中，当用户申请一段存储区域时，系统自动将段的长度折算为若干个页，并为其分配页面。这样，每个段由若干个页组成，这些页可以是不连续的。因此，与段式管理不同，段页式管理的一段原则上不是一个整体的存储区域，而是由若干个分离的存储区组成。为了实现地址映射，系统必须为每个段设立一张页表，记录段中每页的物理页号。同时，系统还有一张段表，段表的表项中记录了隶属该段的页表的起始地址。这样，给出一个相对地址，根据段号就可以找到该段所属页表的入口地址，再根据页号，就可以找到该页对应的物理页号（即该页的存储起始地址），将物理页号与页内地址拼接，可以得到操作数的物理地址，完成地址映射。地址映射的过程如图 2-9 所示。

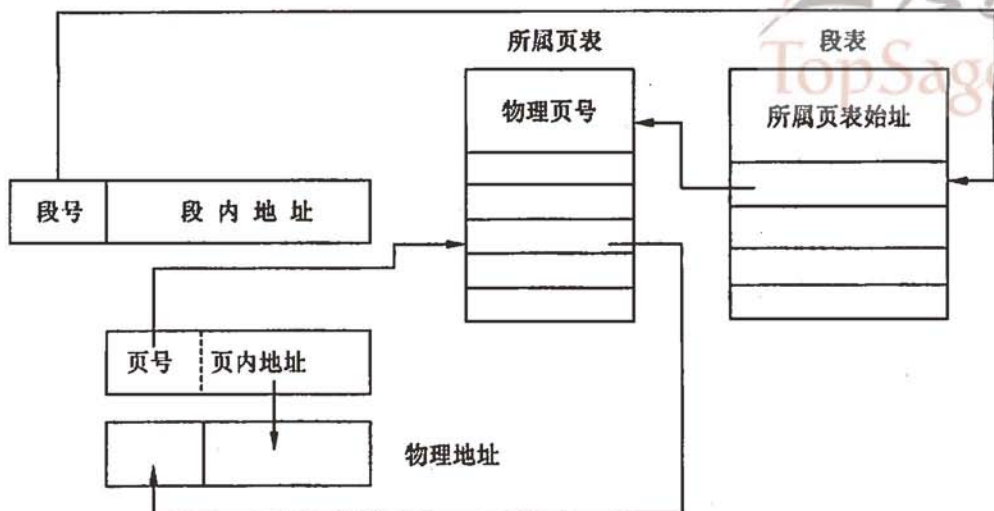


图 2-9 段页式管理地址映射

如果为了进行地址越界检查，在段表的表项中可以再加一项内容，即该段的段长，段表的表项由两项内容构成：段长和所属页表的始址。这样，查到段表的表项后，可以将段内地址与段长比较以确定是否越界。

**【例 2-31】** 简述段页式管理的优缺点。

**解：**段页式管理在用户层面上呈现的是段式管理的行为，比较符合用户的使用习惯；而在实现时内存固定分割，方便管理。当然，段页式管理的硬件支持显然要复杂得多。现代的计算机都在硬件上支持段页式的内存管理，Intel 公司的 CPU 芯片从 80386 开始，就配备了段页式管理的硬件支持装置。

**【例 2-32】** 什么是进程的内存空间？什么是进程的编址空间？

**解：**所谓进程的内存空间，就是进程在实际运行时操作系统分配给它的内存区域，也就是进程可以实际使用的内存大小。所谓进程编址空间，就是进程程序允许使用的最大空间，包括存放程序和运行数据所需的空间，也即进程程序中所能出现的最大地址。在没有虚存机制的系统中，两者是相等的。

**【例 2-33】** 什么是内外存对换技术？

**解：**存储管理中的一个常用的技术是内外存对换技术。操作系统中，有多个进程在系统中同时存在，而进程只有在调度时被选中才能运行。内存的空间是有一定数量的限制的，如何解决使尽可能夺得进程同时存在，而每个进程又可以尽可能多地占有内存空间，比较好的办法是将暂时不能运行的进程放在外存，等到即将恢复运行时再调入内存，就形成了内外存对换。这种技术首先在分时系统中使用。假定系统的内存为 4MB，系统同时接纳 6 个用户进程，如果都放在内存，每个用户进程只能使用 0.66MB 内存，如果只放 4 个，每个进程就可以占用 1MB 内存。进程的运行是遵循时间片轮转的方式的，



因此，内存中只放 4 个用户进程，每个进程占用 1MB 内存。运行进程一旦时间片到，就立即转入外存，同时将在外存的一个进程调入内存。由于磁盘的运行可以与计算在物理上并行，这种对换对计算速度的影响也不大，如图 2-10 所示。

在第二个时间片中，A 和 E 交换了内外存的位置。在第三个时间片中，B 和 F 交换了位置。这样，当第五个时间片时，F 将被调度运行，而此时它早就在内存了。

与对换技术类似的是覆盖技术。这时，原先在内存的部分不需移到外存，只需从外存调入所需要的部分将其覆盖即可。例如，两个进程可以共享一个存储区域用作各自的常量区，在程序运行时，常量区是不会改变的，因此无需对换，简单覆盖即可。

**【例 2-34】** 什么是程序局部性？为什么会产生程序局部性？

解：所谓程序的局部性，包括空间局部性和时间局部性。所谓空间局部性，是指某个地址一旦被使用，在最近的一段时间里，它附近的地址通常也会被访问；所谓时间局部性，是指某个地址被使用，在最近的一段时间里它很可能将再次被使用。导致程序局部性的原因是通常程序中包含大量的循环（这也是计算机可以提高工作效率的主要原因之一），数据结构中又会经常出现数组等存储分配区域比较集中的结构。前者导致变量和代码被重复使用，后者则引起访问区域相对集中。

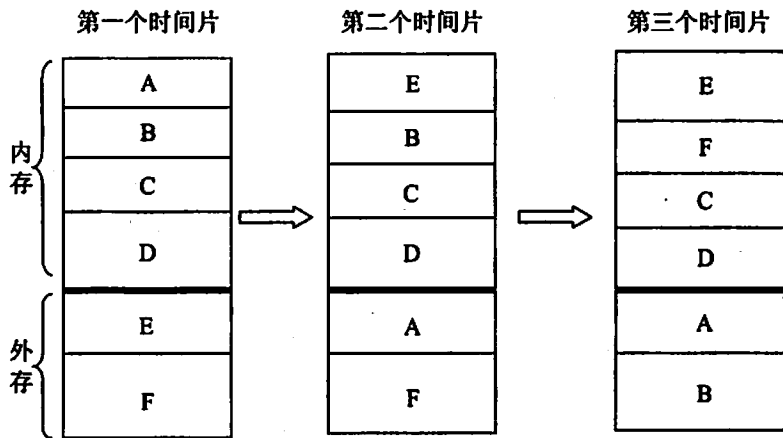


图 2-10 内外存对换示意图

**【例 2-35】** 什么是虚拟存储技术实现思想？虚存得以实施的基础是什么？

解：虚存技术是一种资源转换技术，即将 CPU 资源和外存资源转换为编址空间的技术。

在实现上，配备虚存机制的操作系统允许进程的编址空间将大于进程的内存空间。在外存中，存放了进程运行的全部内容，包括程序和数据。但是，在内存中，只存放了外存副本的一部分内容，通常是程序和部分数据。运行过程中，当操作数已在内存，则正常进行计算，如果发现操作数不在内存，就需要进行对换，将内存中的一部分内

容换到外存，腾出空间换入计算所需的内容。这样，在运行过程中将随时可能出现对换工作。

虚存能够正常进行工作的基础是程序的局部性和多道程序设计技术。前者导致在一段时间内，进程运行的操作对象可能局限在部分区域内，后者则在一个进程进行内外存对换时，系统可以运行另一个进程，CPU 资源不会浪费。

**【例 2-36】** 虚存如何实现？影响虚存效率的主要原因是什么？

解：虚存的实现原理相对比较简单，系统为每个进程在外存中设置一个完整的副本，同时，为其在内存分配固定大小的区域（常常称为页框），放置一部分运行数据。当进程运行中要使用的某个操作对象不在内存，这时计算机将发出中断（称为缺页中断，因为通常对换的基本单位是页），进入操作系统处理。操作系统根据某种算法，确定被对换的对象（通常称为确定淘汰页），将其存回外存，再将该操作对象所在的外存调入刚刚腾出来的内存位置，完成缺页中断的处理。该进程就可以继续运行了。

实际系统中大量使用的是页面调度方法。这里，对换的基本单位是页，它的长度固定，管理比较方便，现代计算机都支持页面调度。无论是页式管理或段页式管理，都可以由页面请求调度机制来支持。但是，段式管理也可以实现虚存，不过，实现的方式相对就要复杂一些，因为对换的要求将更加严格，在选择淘汰段时，不仅要考虑使用、调入、访问等的情况，还必须考虑另一个限制因素，即被淘汰的段必须不小于将被调入的段。

影响虚存效率的主要因素是命中率，因为缺页中断处理涉及外部设备（通常是磁盘）的运行，花费时间更长。

**【例 2-37】** 除了淘汰算法以外，还有哪些程序设计技巧有助于提高系统性能？

解：通常，在页表的表项中增设修改位，页面调入内存时，修改位为零，在该页被修改时置为 1。这样，当未修改过的页被淘汰时，不需回存磁盘，由于缺页中断处理费时的主要环节是内外存对换，这样时间可以减少将近一半。或者还可以采用提前回存的方法，当磁盘空闲的时候，系统有意识地将修改位为 1 的页面存回磁盘，同时将它的修改位清零，以此来提高淘汰页不必回存的概率。根据修改位，还出现了一些变种的淘汰算法。如将 FIFO 变为首先在未修改的页面中选择调入时间最久的页面加以淘汰等。

此外，设置“偷页进程”，当 CPU 不忙的时候，偷页进程运行，预先确定下一个将被淘汰的页面，同时将它预先回存磁盘，这样，有可能在真要淘汰它时就可以不必回送磁盘了。

**【例 2-38】** 什么是颠簸现象？

解：虚存机制运行时，随着缺页中断的频率增大，系统的性能急剧下降，最终会出现“颠簸”（又称“抖动”）。当一个系统用于正常计算的时间少于处理缺页中断的时间时，就认为出现所谓的颠簸现象。令  $P(s)$  为缺页中断发生的概率， $T$  为内外存交换一页的时

间,  $t$  为指令执行的平均时间。当  $T < t/P(s)$  时, 系统是安全的; 当  $T = t/P(s)$  时, 系统处于临界状态; 而  $T > t/P(s)$  时, 系统就会进入颠簸状态。

### 2.2.3 文件管理

文件是操作系统管理的最重要的一类软资源。文件存储在辅助存储器(通常是磁盘, 后备文件也存储在磁带上)中的具有标识的一组信息集合, 除了特别指明外, 它们通常是永久保存的。对文件的分类可以从各种不同的角度。从文件的内部的信息组织结构来分, 文件的逻辑结构可以分为记录文件、流式文件等; 从文件内容的存放与查找方式来分, 文件的物理结构可以分为连续文件、直接文件、链接文件、索引文件、索引顺序文件等。而文件系统则是文件及其管理机构的总称。每个文件都有一个文件控制块, 它是文件存在的惟一标志。为了方便文件的检索, 文件控制块是按照某种规则排列的。文件控制块的有序集合称为文件目录, 文件目录的组织形式体现了文件控制块的排列规则。包括一级目录(顺序表目录)、二级目录、多级目录(通常为树型目录)等。同时, 文件系统要为用户提供多种使用文件的手段。

由于文件通常是存放在磁盘上的, 操作系统除了必须管理磁盘设备以外, 还必须对磁盘空间进行管理。当文件创建时, 需要申请空闲的磁盘块来存放其信息, 在其后文件添加内容时也一样; 当文件删除全部或部分内容时, 需要回收其占用的磁盘块。磁盘空间的管理主要体现磁盘空闲块的管理上, 通常有位图(页示图)法、链接法、索引法和空块链接法等。文件系统提供了一系列使用文件的手段, 包括打开文件、关闭文件、改变文件系统的当前目录(或使用目录)、读文件、写文件、文件定向等。

#### 【例 2-39】什么是文件?

解: 文件是存储在辅助存储器(通常是磁盘, 后备文件也存储在磁带上)中的具有标识(文件名或代号)的一组信息集合, 人们可以通过标识来访问文件。通常, 文件是永久保存的。根据文件存放的信息类型, 文件可以分为文本文件、二进制代码文件、图像文件; 根据用途, 文件可以分为档案文件、程序文件(还可以细分)等。文件名带有后缀, 后缀则表示了文件的类型, 例如.doc 表示 word 文件, .c 表示 C 语言程序文件、.pdf 表示图像文件、.o 表示目标代码文件等。

需要说明的是, 现代操作系统中都将设备的使用在形式上等同于文件的使用, 因此, 在文件系统中还设置了一类文件用以代表设备, 称为特殊文件。通常特殊文件放在/dev 的目录下。

#### 【例 2-40】通常, 文件的内容组织形式有哪两种?

解: 从文件内部信息的组织形式来看, 文件可以分为记录文件和流式文件。所谓记录文件是指文件由若干个记录所组成, 每个记录有固定的格式, 可以用数据结构的形式表示, 同一个文件里所有的记录都具有相同的格式。而且, 记录也是有标识的, 换句话



说可以根据记录的标识来检索记录。在记录文件里,访问信息的单位是记录,要读取某个字段,必须先读出它所在的记录,然后再访问它。所谓流式文件是指文件内部没有格式,把文件看做是一个串,就好像是一篇没有章节、没有段落的文章,只有长长的一串文字。当然文件的内容肯定是有语义层次的,对文件内部的信息结构交由用户自己来解释。这种方式比较灵活,目前常用的操作系统如 UNIX、Windows 都提供流式文件。流式文件又分为字符流文件和二进制流文件两种。当然,流式文件也有页的概念,但它与其说是一个逻辑概念,还不如说是一个物理概念,文件在磁盘上当然是分块存放的,每一块就是一页。

**【例 2-41】 简述文件物理组织的分类。**

解:文件的物理组织是指文件各部分之间的联系与在磁盘上存放的方式,可以有多种。

连续文件是指文件的内容在磁盘介质上是集中、连续存放的,按照文件内容的顺序存放在连续的磁盘块上,只需要知道文件的起始地址和长度就可以访问文件。显然,这种文件整体存取的速度很快,但要对文件内容进行增、删、改就相当麻烦。适宜于存放相对不变的文件,例如,系统文件的存放就用这种形式。

链接文件按文件内容的次序分为若干块,存放在不同的磁盘块上,前一块中指出后一块的磁盘地址。形式上,文件就像链表一样。因此,只要知道文件第一块的地址,就可以依次读取文件。显然,它的优缺点恰好与连续文件相反,文件的增删改相当容易,只要修改地址指针即可,但成批读写效率就太低了。

索引文件就比较灵活,文件系统为每个文件设置一张索引表,索引表的表项依次记录每个文件块的磁盘地址,通过查询索引表,就可以知道该文件块的存放位置,从而方便地启动磁盘访问每块文件的内容,增加的开销就是索引表。

索引顺序文件与索引文件类似,不过就是文件块也按页号次序顺序存放,对页号采取索引管理。同时要注意处理溢出问题。

直接文件又称 Hash 文件,比较适合记录文件。依据某个散列算法,通过记录名直接对应存放地址,不过要注意冲突问题。

**【例 2-42】 简述索引的种类。**

解:索引文件又可以分为直接索引和间接索引两类。在直接索引方式,索引表中的每一项内容就是文件块的磁盘地址,每个文件只有一张索引表,文件不断增长时,索引表也在不断加长,当索引表长度超过一个磁盘块时,就用链接方式增加一块索引。这样,文件总体上是索引的,但索引表本身又是链接的。间接索引又叫多级索引,这时,只有最后一级索引是直接索引,即最后一级索引表中每一项的内容直接指示文件块的磁盘地址;而前几级索引表的每一项内容指示的是下一级索引表的地址,图 2-11 所示是一个二级索引的例子。

但是,多级索引有时也不太灵活,因此,有的系统设置了混合索引,即一个文件的



索引中既包含直接索引,也包括多级间接索引。例如在 UNIX 中,放置在文件控制块中的索引表共有 13 项,前 10 项是直接索引项,每一项内容指向文件块的磁盘地址;第 11 个是一级索引项,指向一张一级索引表(后者的每个表项指向文件块);第 12 个是二级索引项,指向一个二级索引表(后者的每一个表项分别指向一级索引表);第 13 个是三级索引项,指向一个三级索引表。假定一个索引表可以存放 256 个地址,那么 UNIX 文件的最大长度可以为  $10+256+256^2+256^3$ ,已经足够大了。显然,索引文件,特别是混合索引文件,很适合随机存取和文件动态增长。以 UNIX 为例,连同读取文件控制块在内,读取任何一页文件,启动磁盘的次数不大于 5 次。如果使用直接索引,读第 2570 页就需要启动磁盘 10 次以上。如果使用链接文件就要 2000 多次!

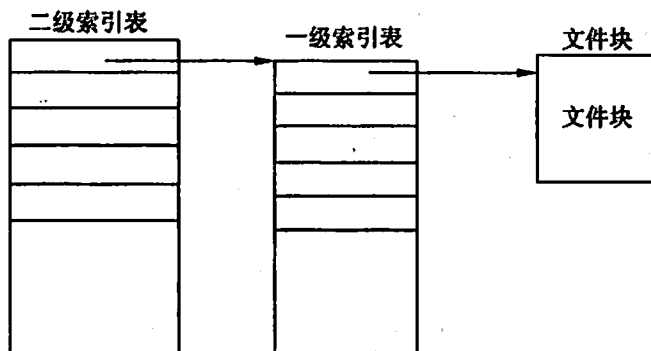


图 2-11 二级索引示意图

**【例 2-43】** 什么是文件控制块 (FCB)? 什么是文件目录?

**解:** 文件控制块是文件系统中最重要的数据结构,是文件存在的惟一标志。它存放文件的一些基本信息,主要包括三大部分:文件的标识信息,包括文件名、所有者名、文件类型、文件最近修改时间等;文件的位置信息,包括文件的长度、文件存放位置(例如,对索引文件来说就是索引表地址、对连续文件来说就是文件存放的起始地址、对链接文件来说就是第一个文件块的地址等);文件的访问权限信息,例如口令、保存时限、保护类别等。一旦创建文件,文件系统首先为其生成一个文件控制块。

文件目录是文件控制块的有序集合,将系统中所有的文件控制块按照某种规律组织起来以便于检索,就形成了文件目录。必须注意,文件目录也由文件组成。

**【例 2-44】** 简述文件目录的组织方式。

**解:** 通常,文件目录分为一级目录和多级目录。所谓一级目录,就是将文件控制块排列成一张顺序表。显然,这种情况下文件的平均检索长度为表长的一半,很费时间。直接文件也是一种一级目录,不过它利用 Hash 算法建立文件名与其文件控制块在表中的序号的对应关系,以减少检索时间。多级目录中,IBM 首先使用的是二级目录,第一级目录是用户目录,组成一张顺序表,其表项为用户名和相应的文件控制块表地址。第二级目录为按用户区分的文件控制块表,每个用户一张顺序表,存放他自己的文件控制

块。这样，可以比一级目录大大缩短检索时间。例如，系统中有 10 个用户，每个用户平均拥有 10 个文件，用一级目录管理，文件的平均检索时间为 50，而用二级目录，平均检索时间将缩短为 10。

现在最常用的多级目录是树型结构目录，UNIX、Windows 等都使用树型目录。树型目录中，第一级为根目录，常驻内存。文件分为普通文件和目录文件两类，树中所有的非叶结点都是目录文件，只有叶结点才是普通文件。目录文件是一张顺序表，表中每一项记录它的直属下级文件名和相应文件控制块的存放位置，如图 2-12 所示。

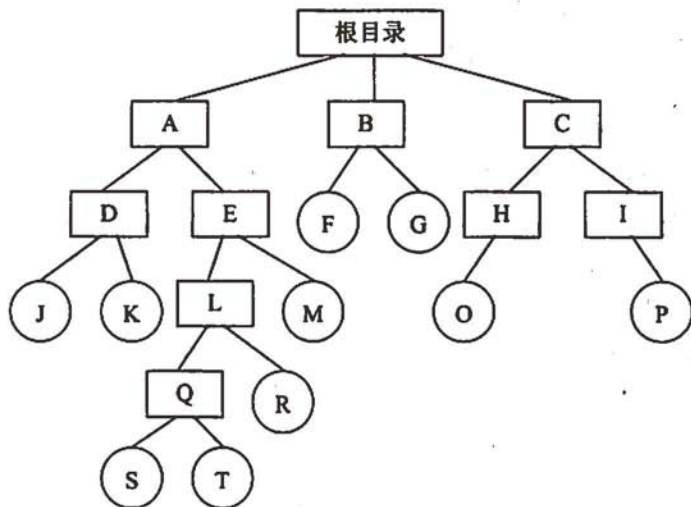


图 2-12 树型结构目录

**【例 2-45】** 简述文件的路径名。什么叫做工作目录或当前目录？

**解：**文件的全路径名描述了从根目录到该文件的周游路径。以图 2-12 为例，普通文件 S 是目录文件 Q 的直属下级，Q 还有其上级，依次向上，形成的全路径名为：/A/E/L/Q/S，其中第一个“/”表示根目录。利用全路径名检索文件时要从根目录开始，显然比较繁琐。因此引入部分路径名，一般文件系统都设置了工作目录（又称当前目录），用户可以自行指定当前目录。一旦某个目录文件被指定为当前目录，它就被调入内存。部分路径名描述了从当前目录到文件的周游路径。例如上例中指定 L 为当前目录，则文件 S 就可以用它的部分路径名来检索 Q/S，如果当前目录是 Q，直接用 S 即可。显然，同一个目录下的文件不能重名，不同目录下的文件是可以重名的。在 UNIX 中，还可以对文件进行连接，就是允许一个目录下的文件也成为另一个目录的直接下属。这时，文件目录就不再是树型，而是偏序的了。

**【例 2-46】** 简述为什么在使用文件前必须打开文件，使用完毕后必须关闭文件。

**解：**通常文件控制块是存放在磁盘上的，打开文件操作的结果就是将文件控制块调入内存，还为其开辟一个缓冲区。操作的方式分利用系统调用和命令两种，系统调用嵌

入程序中运行, 执行打开文件操作结果一般还会返回一个文件临时的返回值 (正整数), 在文件的本次打开到关闭的期间, 返回值即代表该文件, 它的含义是该文件在打开文件表中的表项序号, 用户以后就使用它来操作文件, 直到该文件关闭。如果再次打开, 有可能返回不同的返回值。将 FCB 调入内存的同时, 还将文件内容调入内存缓冲区。系统设有关闭文件的系统调用, 关闭文件则将首先将缓冲区的内容存回文件, 再将文件控制块存回内存, 清空打开文件表相应的表项。通常, FCB 中还设有文件读写指针, 指向文件当前访问的位置, 文件刚打开时为 0。系统设有改变文件读写指针的函数, 而在联机方式, 移动鼠标就改变了它的值。在联机方式, 打开文件则是鼠标双击代表文件的图符, 关闭文件就是单击代表关闭的图符。

## 2.2.4 设备管理

外围设备是计算机操作系统管理的一类重要资源, 与 CPU、内外存一样, 属于硬件资源。用户使用外围设备, 都必须通过操作系统进行。实际上, 用户只是向操作系统提出设备服务的请求, 由操作系统根据用户的要求, 控制外围设备来完成输入/输出的实际活动。因此, 设备驱动程序只能由操作系统来驱动。

外围设备的主要任务是实现输入输出, 达到与外部交换信息的目的。如果把外部世界看做一个大文件, 在这一点上与读入文件内容和向文件写入内容形式上雷同。因此, 现代操作系统往往将设备管理与文件管理统一起来, 用一个专门的目录来存放设备的驱动程序。通常在根目录下设置一个 dev 目录, 每一个设备对应 dev 目录下的一个子目录, 每个子目录下设置若干个文件, 分别对应该台设备的各类操作, 如读、写和控制操作等。

当然, 由于设备毕竟不是传统意义下的文件, 因此, 我们必须讨论与设备有关的一些特殊管理策略, 如设备与主机的连接、设备的控制、设备之间的模拟, 以及设备的调度等等问题。

### 【例 2-47】简述设备的分类策略。

解: 外部设备的分类可以从不同的角度进行。

从设备交换数据的方向来看, 可以分为输入设备、输出设备与混合型设备 (兼有输入/输出)。

从设备与主机交换数据的形式来看, 外设可以分为字符设备和块设备两类。所谓字符设备, 是指以字符为单位与主机交换数据的设备, 例如打印机、键盘、显示器等。显然这些设备交换数据的速度较慢, 又称为慢速设备。所谓块设备, 交换数据是以块为单位的, 例如磁盘, 交换数据的单位是一个分区或扇面, 显然它的速度较快, 又称快速设备。磁盘、磁带等都属于快速设备。

从使用特点来分, 外设可以分为独占设备和共享设备两类。所谓独占设备, 是指在使用时适宜于在某个程序运行期间内由它独自使用, 如果在该程序运行中改变分配



关系,交给其他程序使用,就会出现混乱。例如打印机,程序在运行过程中可能需要不时地输出运行结果,如果几个程序共享,打印结果将难以分清,键盘、显示设备等都属于独占设备。一般而言,独占设备是静态分配的,程序运行前分配设备,直到运行结束后收回,当然这样它的利用率较低。所谓共享设备,则允许多个程序在运行期间交替地使用设备,不会引起混乱。例如磁盘,一次访问结束后,就可以由另一个用户使用,在适当的时候,前者又再次使用,不会引起任何问题。共享设备则系统随时进行分配和回收。因此,必须保证设备正确的互斥使用问题,同时需要精心考虑调度算法以提高设备的使用效率。

**【例 2-48】** 简述计算机控制设备进行数据传输的方式。

解:设备与计算机之间数据交换的控制方式主要有三种:程序控制、I/O 中断和 DMA (直接存储访问)。程序控制方式通过由 CPU 执行程序启动数据传输,然后等待传输的完成,而在数据传输期间,CPU 并不运行别的程序;I/O 中断则由 CPU 启动数据传输,然后 CPU 转去执行别的任务,传输结束后设备发出中断,提醒操作系统作设备结束处理,利用通道控制外部设备就属于这种情形;DMA 则在内存和设备间开辟直接的交换通路,无需处理器的干预。为了加快速度,通常还使用 I/O 缓冲区,协调快速的处理器和慢速的设备的速度,这种方式就是目前的输入/输出板卡的工作方式,说到底,在策略上与通道是类似的。

**【例 2-49】** 简述计算机与设备的连接和控制方式。

解:这个问题与上面的问题类似,不过更加深入一些。

主机对外部设备控制总的可以分为两大类:

程序控制方式。主机执行程序,控制外设启动数据传输,通过设备寄存器直接控制设备和数据传输操作。这种方式主机的负担较大,不利于计算与外设的并行。而且由于两者之间在速度上的巨大差异,造成主机的等待和浪费。这种方式设备是不能独立工作的,在早期曾经使用。

设备独立操作方式。这种方式下 CPU 不再直接控制设备的数据传输,主机启动输入/输出操作以后,设备在控制器的控制下独立工作,主机将可以转去运行别的程序。只有在一些需要主机干预的时刻,例如 I/O 结束、纸张用尽、操作有错、数据就绪等。设备发出中断,主机接收中断、操作系统做相应中断处理。设备独立操作目前主要有两种机制:通道机制和智能接口控制机制。

通道机制主要在大中型计算机中使用。所谓通道,是一种特殊的处理机,专门用来处理 I/O。主机只要为每次 I/O 准备好通道程序,然后启动通道,后者就自行执行通道程序,控制具体设备进行操作,通过 I/O 总线直接完成设备与主存之间的数据交换,在 I/O 结束时发出设备结束中断。通道有两种基本的类型:多路通道和选择通道。多路通道可以分时地执行多个通道程序,同时控制多台设备的工作,执行通道程序的单位又称为子通道。其中,多路通道又可分为字节多路通道和数组多路通道,前者可连接多台慢速设



备同时传输，后者可连接多台高速设备，但其中一台传输时，其他只能进行控制操作。选择通道在一个时间内只能执行一个通道程序，用来控制高速设备。

智能接口机制在微、小型计算机中使用。计算机通过 I/O 总线与外部设备连接，在 I/O 总线上插有含有 I/O 控制器乃至 I/O 处理器的接口卡，它接受 CPU 的委托和激发，控制设备完成 I/O 操作。近年来，控制器可以拥有大量的存储空间，处理能力日益扩大。总线的标准化和开放性也日益发展。

设备的驱动是通过执行驱动程序实现的，完成设备初始化、设备中断处理和数据传输操作，由操作系统或 I/O 处理机制运行。它向上与高级的 I/O 原语对应，向下与设备硬件对应。

**【例 2-50】** 大型计算机中，CPU 与设备的连接是如何实现的？

解：在大型计算机系统中，计算机与设备的连接分为三层：通道、控制器和设备，见图 2-13。

图 2-13 中的多路连接技术，可以保证主机与设备有多条连接，有效地避免出现瓶颈，或因某些装置发生故障造成设备无法运行。

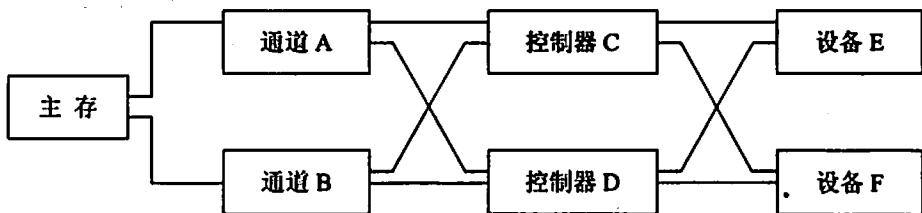


图 2-13 计算机与设备连接图

**【例 2-51】** 设备管理的主要工作有哪些？

解：设备管理主要有分配设备、回收设备、输入、输出等。对具有控制功能的设备，如磁带机，则还有回绕、向前等。

系统为每台设备编号，设备编号分为主设备号与次设备号，主设备号用来区分设备的类型，次设备号表示同类设备中的序号，每台设备都有这两个号，两个号码合在一起就惟一确定了一台设备。系统建立了一张设备表，每台设备占一行。进程申请要求分配设备时，必须指明主设备号。系统接到申请后，在设备表里主设备号相符的行中挑选使用栏为空的行，在该行的相应栏中填入申请进程的进程号，并将该设备的次设备号返回进程，进程就可以使用了。进程释放设备时，系统根据主、次设备号确定了设备行，将使用栏清空就完成了回收。

进程使用设备进行输出操作时，系统根据给出的参数，将输出数据进行加工（如格式加工、二进制转为十进制等）调用相应的驱动程序；进行输入操作时，完成相反的操作，将内容存到用户指定的地址。

在现代操作系统中，将设备管理纳入文件管理一类，从而统一了这两种系统资源的管理方式。

**【例 2-52】 什么是虚设备技术？**

解：设备管理的一个重要问题是如何提高设备的利用率以及使独占设备共享化。为此，引入了“虚设备”技术。所谓虚设备技术，是指用一类设备（通常是高速设备）来模拟另一类设备（通常是低速设备）的技术，被模拟的设备称为虚设备。例如，在批处理系统中常常使用的脱机输出技术。打印机是一类独占设备，适宜归一个用户使用。但是，一个批处理系统中可能存在许多用户，每个用户都需要输出自己的计算结果，但每个用户都只是在运行期间偶尔地、间断地打印，为每个用户配一个硬件打印机无疑是不经济的，而共用一台打印机又会使打印结果无法区分。为此，操作系统可以为每个用户在磁盘上设立一个打印文件，将它作为模拟的打印机分配给用户。当用户程序执行打印的系统调用时，并不直接启动打印机打印，而是将打印内容写入相应的输出文件。在用户程序运行结束后，添加封面，加上输出文件的内容一并打印出来。这样，用户感觉自己独占了一台打印机，可以随时输出计算结果，但实际上只是在运行结束后才使用物理打印机一并打印，分给它的是一台虚拟打印机。既提高了打印机的使用效率，又取得了很好的效果。这是在操作系统的管理下实现资源转换的又一例子。当然，这时用户不能联机地检查自己的输出结果，只能用于批处理情况。

同类设备也可以模拟，例如多窗口技术就是显示器模拟自身的例子，使一个屏幕同时监控多个进程的运行情况。

## 2.2.5 作业管理

作业是操作系统管理的一类重要的软资源，分为脱机作业和联机作业两类。所谓联机作业，是其运行过程中一直受到用户控制（或用户可以控制）的作业。以一个作业的运行为例，用户控制源程序和数据文件的输入，为它们指定文件名；然后，用户调用编译程序对源程序进行编译，将其存入指定的文件中；接着，用户调用连接装配程序，将若干个编译成目标代码的程序进行连接装配，形成可执行程序，存入又一个文件中；最后，用户运行连接装配好的程序，得到所希望的运算结果。显然，联机作业的控制完全取决于用户，控制过程用户可以随时改变，甚至在作业运行用户也可以随时中止。而脱机作业则不一样，用户对作业的运行过程没有任何控制权，用户必须事先设计好作业运行的控制过程，书写作业控制说明文件，连同源程序文件和运行数据文件一起交给系统操作人员，由操作人员输入计算机系统，系统根据作业控制说明自动控制作业的运行，得出运行结果，由操作员交给用户。脱机作业的运行必须有批处理操作系统的支持，所以又叫批处理作业。

**【例 2-53】 什么是作业的脱机控制？何谓 Spooling 系统？**

解：作业的控制可以分为两类：脱机控制和联机控制。所谓脱机控制就是指作业一旦提交，用户本身就不再对作业的运行有任何的控制权，完全由操作员将其装入系统。作业运行的过程中，完全根据作业事先安排的步骤一步一步地运行，直至运行结束。在运行过程中，用户、操作员都不做任何干预。运行结束后，系统将作业运行过程中的输出数据按照统一的格式组织并输出。作业的脱机控制可以提高系统资源的使用率。在早期，作业的输出、运行结果的输出都是由专门的计算机来完成的，称为前置机和后置机。这时，一个脱机作业系统由三台计算机组成。多道程序技术出现以后，操作系统可以利用一个进程来控制作业的输出（称为预输入进程），另一个进程控制作业的输出（称为脱机输出进程）。典型的脱机作业管理系统就是 Spooling 系统，即外围设备同时操作的联机工作系统。联机系统则由用户自始至终对作业进行控制，从作业的输出、作业运行过程的每一个步骤、直到作业的输出和退出，都由用户通过命令（包括键入命令行或鼠标单击图符）加以控制。就像我们通常在 PC 机上工作的那样。

【例 2-54】 简述脱机作业的生命周期，其各个阶段是如何控制的？

解：系统将作业的生命周期分为四个阶段：进入阶段、调度阶段、运行阶段和撤离阶段，对应地，作业也有进入、后备、运行和撤离四个状态。分别由四个进程加以控制。从作业进入系统起，其状态的转变及其过程都由系统加以控制，用户与操作员均无需干预。图 2-14 清楚地显示了作业生命周期的全过程。

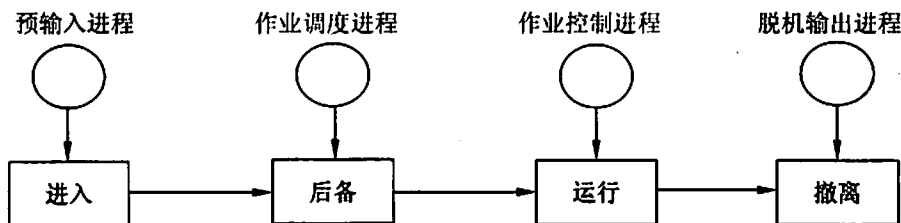


图 2-14 作业生命周期图

【例 2-55】 简述脱机作业的组成，何谓作业控制说明。

解：为了实现系统的自动控制，作业必须提供足够的信息。因此，脱机作业由两个部分组成：作业说明书和作业体。作业说明书包括两部分内容：作业描述与作业控制说明，其中作业描述提供了作业的基本描述信息，例如作业名、用户名、作业运行的资源要求、优先数、估计运行时间、完成时间限制等。通常用结构化英语书写；作业控制说明详细列出了作业运行的控制步骤（称为作业步）以及例外处理安排，用作业控制语言书写。作业控制语言是一种特殊的语言，用来刻画作业运行的控制过程，它的文体可以有二种形式，一种表现为命令行，每一行由操作符、操作对象和参数组成；另一种则和通常的程序设计语言类似，形成一段程序，例如 UNIX 的 Shell 语言。系统通过对作业



控制说明的解释执行，完成作业的运行。而作业体则包括程序和使用的数据。

**【例 2-56】** 什么是作业控制块？简述其组成。

解：脱机作业系统的一个重要的数据结构是作业控制块（JCB），JCB 是作业存在的惟一标志，刻画了作业的基本信息，系统通过对作业控制块的管理来管理作业。JCB 的内容通常包括作业名、用户名、优先数、作业状态、资源要求、作业控制说明文件名、作业体文件名、脱机输出文件名等，在作业进入过程中生成。

作业状态是批处理系统一个特有的名词，作业进入系统以后，生成作业控制块。但这时作业并不立即运行，而是进入输入井等待被调度，此时作业处于后备状态；作业被调度开始运行后，作业状态改为运行状态；运行结束后，须将其运行结果进行脱机输出，此时作业处于撤离状态，一旦输出完成，作业就不复存在了。

作业控制块是在作业进入时由系统生成的。

**【例 2-57】** 简述脱机作业在系统中的变化过程。

解：作业的进入是由预输入进程控制的。预输入进程控制输入设备，将作业读至内存，从作业说明中分离作业控制说明，形成单独的文件加以存放，将作业体也作为文件加以存放，同时根据作业说明书填写 JCB，并置为后备态。需要说明的是，系统在外存开辟专门的区域存放后备作业，这个区域称为预输入井。在预输入进程的控制下，作业源源不断地进入预输入井，形成后备作业集合。

作业调度进程的任务是挑选符合条件的作业为其准备运行条件，包括为其分配资源、将其作业控制说明文件调入内存。它依据某种策略，检查后备作业控制块，选中符合条件的作业，将其状态改为运行态，并将相应的作业控制块交给作业控制进程，自己返回去继续调度工作。如没有作业控制进程接收则等待。由于作业调度的影响时间区间是整个作业的运行时间，因此又叫远程调度或高级调度。

作业控制进程在系统初始化结束以后，向作业调度进程要求 JCB，如没有则等待。一旦接到 JCB 就控制作业的运行，它解释执行作业控制说明，完成作业的运行。所以，作业实际上是“嵌入”在作业控制进程中运行的，与其说作业控制进程是在运行作业程序，倒不如说它在运行作业控制说明更贴切，前者的运行是后者运行的必然结果。作业运行结束后，作业控制进程将当时的 JCB 转交给脱机输出进程，自己返回向作业调度进程继续索要 JCB。

脱机输出进程在系统初始化结束以后，向作业控制进程要求 JCB，如没有则等待。一旦接收来自作业控制进程的 JCB 后，调出相应的脱机输出文件，按照固定的格式打印。然后归还其占有的资源，完成作业的撤离工作。然后返回去继续向作业控制进程要求 JCB。

需要说明的是，脱机作业系统往往还配备若干任务进程，这时还将有一个任务调度进程。由于一个作业的不同任务有些是可以并行的，这时作业控制进程将可以并行的任



务交给任务调度进程，由它调度某个任务进程加以执行。显然，任务调度的影响时间区间是任务的执行时间，所以又叫中程调度或中级调度。

**【例 2-58】** 简述作业调度算法的种类并加以评述。

解：在脱机系统中，人们最关心的是系统的吞吐率。作业调度策略的一个重要目标是提高系统的吞吐率，同时，又不要造成某些作业陷入“无限等待”。作业调度算法主要可以分为基于优先数和基于资源利用两类。前者的优先数可以分为静态、动态两种，静态优先数在作业生成时一次确定，既可以直接由用户给出，也可以根据用户的要求，结合作业本身的特性（如对资源的要求等）来确定，确定以后不再改变；动态优先数则表示在作业进入后，优先数可以随时间的推移而改变。特别是动态优先数法，如何确定优先数本身就体现了调度目标的考虑。后者则着眼于资源的合理和统筹使用，尽量减少作业之间对资源整体上的竞争。这样同样可以达到可以提高资源的利用率和提高系统吞吐率的目的。

给予优先数的算法主要有以下几种。

(1) 先来先服务

按作业到来的次序调度运行。显然不考虑吞吐率问题，作业到来时间就可以视为优先数。

(2) 直接法

优先数由用户指定，或者参考用户的要求和作业本身的特性来给出。

(3) 短作业优先

挑选估计运行时间最短的后备作业投入运行，数学上可以证明，这种算法可以得到最短的平均周转时间（作业周转时间是指作业进入系统到运行结束的时间段，包括后备时间和运行时间），从而得到最大的吞吐率（平均周转时间的倒数）。但是，由于允许“加塞”，在较短作业源源不断到来的情形，长作业将可能在（无法预计的）很长的时间内得不到运行，引起长作业“无限等待”。这时，估计运行时间就是优先数。

(4) 最高响应比优先

这里把响应比作为优先数。 $\text{响应比} = (\text{后备时间} + \text{估计运行时间}) / \text{估计运行时间}$ 。系统挑选响应比最大的后备作业投入运行。显然，当作业估计运行时间相同时，它就是先来先服务；当作业几乎同时到达时，它接近于短作业优先。而且，随着后备时间的延长，作业的优先级别将不断提高。这样，这个算法既可以保持短作业优先的特性，又可以有效地避免“无限等待”的现象。

在多道系统中，有多个作业控制进程，系统中有多个作业处于运行状态。这时，资源搭配算法将显示出很大的优越性，例如，可以把需要较多 CPU 资源的作业和需要较多输入输出的作业放在一起运行，可以各得其所。但是，这要求资源要求必须在作业说明书中仔细列出。这种调度方法称为资源搭配算法。例如，可以将作业根据对各类资源的要求程度分为几个队列，调度时在各个队列之间轮转。

**【例 2-59】** 什么是联机作业系统？简述其对作业的管理和运行方式。

**解：**联机作业系统是允许用户在控制台上自行控制作业运行的系统。一般的分时系统和 PC 操作系统都属于联机作业系统，用户通过命令（包括键入命令和单击命令）控制作业的运行。联机作业系统由一个用户管理模块，记录允许使用系统的用户名及其口令，同时还有一系列记录用户使用系统资源的表格项（如 CPU 使用时间、终端使用时间、打印数量等），以便计算并收费。系统有一个监控进程，它监控每一个与主机连接的终端，一旦终端电源打开，它就要求用户进行系统登录。用户成功登录后，系统为每一个登录的用户生成一个用户进程，用户进程启动后等待接收来自用户的命令。命令的 Enter 键或单击鼠标将产生一个中断，激活用户进程，它根据命令的内容，调用相应的服务例程运行，完成要求的任务，如此循环往复。键盘上还有一个特殊的键（Esc），只要按下此键，将发出中断，正在运行的例程将被终止，用户进程回去等待新的命令。所谓后台运行命令，是指用户进程接到该命令后，将生成一个子进程，由它调用服务例程完成服务，自己则返回等待新的命令，不必等服务完成。例如，在 UNIX 系统中，在正常命令后加一个 &，就变成了后台命令。

此外，用户在键盘上并不一定只能键入单个命令，也可以键入一个有数个命令组成的命令串。在 UNIX 中，数个命令之间用“|”隔开，表示上一个命令的输出就是下一个命令的输入，从而依次执行。

在联机系统中，性能问题主要体现在命令的响应时间长短，所谓响应时间是指命令键入时间到服务完成时间之间的间隔。

## 2.3 思考练习题及答案

### 思考练习题

1. 选择题，从表中选取合适的答案填入\_\_\_\_中。
  - A \_\_\_\_是批处理操作系统中负责将后备作业投入运行的进程。
  - B 操作系统通过检查\_\_\_\_来分析中断的原因。
  - C 通道出现故障，可以通过检查\_\_\_\_来发现原因。
  - D 磁盘属于\_\_\_\_的一种。
  - E \_\_\_\_是整理就绪进程队列的依据。
  - F 同一个信号量上的 P、V 操作必须\_\_\_\_执行。
  - G 打印机是\_\_\_\_的一种。
  - H \_\_\_\_是文件控制块的有序集合。
  - I 虚存中的命中率既有命中内存的一层意义，又有命中\_\_\_\_的另一层意义。
  - J 当资源图上出现回路时，就出现了\_\_\_\_。

供选择的答案:

- |            |          |          |          |
|------------|----------|----------|----------|
| A ① 预输入进程  | ② 脱机输出进程 | ③ 作业控制进程 | ④ 作业调度进程 |
| B ① 中断源    | ② 中断向量   | ③ 中断屏蔽   | ④ 中断位置   |
| C ① 通道状态字  | ② 通道控制字  | ③ 通道接口   | ④ 通道硬件   |
| D ① 块设备    | ② 虚设备    | ③ 字符设备   | ④ 独占设备   |
| E ① 作业调度算法 | ② 任务调度算法 | ③ 磁盘调度算法 | ④ 进程调度算法 |
| F ① 同步     | ② 异步     | ③ 互斥     | ④ 立即     |
| G ① 块设备    | ② 快速设备   | ③ 字符设备   | ④ 共享设备   |
| H ① 文件索引表  | ② 文件地址   | ③ 文件链接   | ④ 文件目录   |
| I ① 磁盘     | ② 联想寄存器  | ③ 文件     | ④ 缓冲区    |
| J ① 无限等待   | ② 抖动     | ③ 死锁     | ④ 死机     |

2. 从资源管理的角度出发, 操作系统可以分为 A、B、C、D 以及 E 五个部分。进程是一个 F 的概念, 而程序是一个 G 的概念, 一般地说, 进程可以看成是程序的一次 H。

供选择的答案:

- |            |          |         |          |
|------------|----------|---------|----------|
| A ① 并发管理   | ② CPU 管理 | ③ 信号量管理 | ④ 中断管理   |
| B ① 虚存管理   | ② PCB 管理 | ③ 进程管理  | ④ 存储管理   |
| C ① 设备管理   | ② PSW 管理 | ③ 虚设备管理 | ④ 共享管理   |
| D ① 文件目录管理 | ② FCB 管理 | ③ 文件管理  | ④ 索引文件管理 |
| E ① 缓冲区管理  | ② JCB 管理 | ③ 链表管理  | ④ 作业管理   |
| F ① 动态     | ② 静态     | ③ 管态    | ④ 目态     |
| G ① 动态     | ② 静态     | ③ 管态    | ④ 目态     |
| H ① 唤醒     | ② 切换     | ③ 运行    | ④ 实例化    |

3. 在传统的操作系统中, A 是系统的独立运行单位, 也是独立的 B 分配单位; 在微内核结构的操作系统中, C 提供了一个运行环境, D 是它的一个组成部分, 可以 E 运行。在传统的操作系统中, 进程之间的通信可以通过使用共享 F 或 G 来实现, 而在微内核结构的操作系统中, 通信是通过 H 机制来实施的。MS-DOS 是 I 的操作系统, Windows NT 是 J 的操作系统。

供选择的答案:

- |         |      |      |      |
|---------|------|------|------|
| A ① 线程  | ② 进程 | ③ 程序 | ④ 核心 |
| B ① 缓冲区 | ② 文件 | ③ 资源 | ④ 并行 |
| C ① 线程  | ② 进程 | ③ 程序 | ④ 核心 |
| D ① 线程  | ② 进程 | ③ 核心 | ④ 程序 |
| E ① 互斥  | ② 串行 | ③ 交叉 | ④ 并行 |

- F ① 缓冲区      ② 外围设备      ③ 优先数      ④ CPU  
 G ① 并行运行      ② 串行运行      ③ 消息传送      ④ P、V 操作  
 H ① 权能      ② 端口      ③ 环境      ④ 协调  
 I ① 微内核结构      ② 管程结构      ③ 传统      ④ 多用户  
 J ① 微内核结构      ② 模块结构      ③ 多任务      ④ 单用户

## 4. 填空题

- A 保留现场必须按照\_\_\_\_的次序进行。  
 B 作业控制说明使用\_\_\_\_来书写。  
 C 最先适配的空块分配算法是指\_\_\_\_。  
 D 进程控制块的内容主要包括\_\_\_\_两大部分。  
 E 段式管理中操作数的相对地址包括\_\_\_\_两部分。  
 F 安全状态是指\_\_\_\_。  
 G 动态优先数是指\_\_\_\_。  
 H 分时系统的基本特点是通过\_\_\_\_来为每个用户服务。  
 I 批处理系统主要追求\_\_\_\_。  
 J 进入操作系统的惟一手段是\_\_\_\_。

5. 在一个虚存系统中, 进程的内存空间为 3 页, 已开始内存为空。有以下访页序列: 1465345254351241。分别计算缺页次数:

- A 使用先进先出的页面淘汰算法  
 B 使用优化算法  
 C 使用最近最少使用算法

6. 一个有两个作业管理进程的批处理系统, 作业调度采用最高响应比优先的算法, 进程调度采用基于优先数(优先数大者优先)的算法。作业序列如表 2-2 所示。

表 2-2 作业序列表

作业名	到达时间	估计运行时间	优先数
A	10:00	50 分	5
B	10:20	60 分	7
C	10:50	40 分	3
D	11:20	80 分	8
E	11:40	30 分	6
F	12:00	70 分	9

计算每个作业的完成时间。

7. 进程 P 通过缓冲区 K 向进程 Q 传送信息, 进程 Q 处理后通过缓冲区 T 向进程 S 传送信息, 进程 S 将信息打印出来, K 和 T 大小一样, 一共传送和打印 N 次, 要求打印



次序与原来次序一样。请用 P、V 操作正确实现之，并尽可能保证并行。



8. 某操作系统的文件系统中，文件控制块中的地址信息有 14 个地址。其中，前 10 个地址是直接索引地址，每个指向文件的一页；第 11 个地址是一级索引地址，指向一级索引表；第 12 个地址是二级索引地址，指向二级索引表；第 13 个地址是三级索引地址，指向三级索引表；第 14 个地址是四级索引地址，指向四级索引表。每个磁盘块存放一个文件页，文件索引表占一个磁盘块。假定每个磁盘块长 1024 字节。这个文件系统允许文件的最大长度是多少？

9. 某操作系统使用的磁盘每个磁盘块长 512 个字节，它的目录文件最多允许有 511 个下级文件，最多存储在两个磁盘块上，目录文件采用链接方式。根目录全部常驻内存。普通文件的 FCB 与 UNIX 系统相同。目前，实际的文件系统如图 2-15 所示。

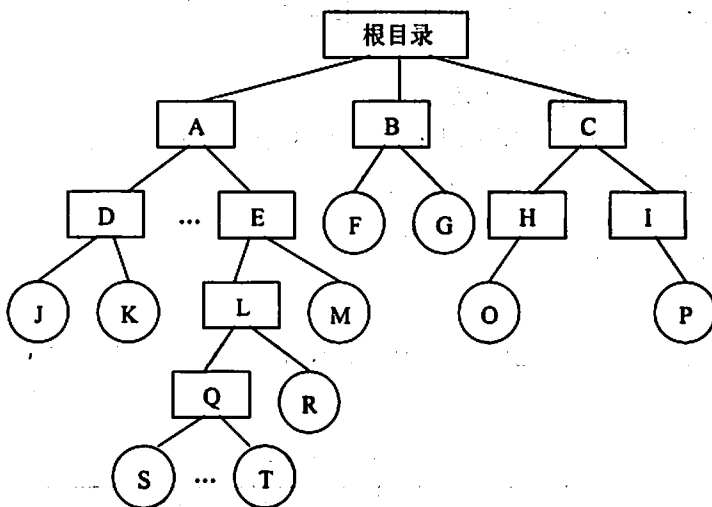


图 2-15 文件系统目录示意图

如果要访问文件 T 的某页，问：最少启动磁盘几次？最多几次？

10. 举例说明，为什么同一个信号量上的 P、V 操作必须互斥执行。

## 思考练习题答案

1. A: ④; B: ②; C: ①; D: ①; E: ④; F: ③; G: ③;  
H: ④; I: ②; J: ③。

2. A: ②; B: ④; C: ①; D: ③; E: ④; F: ①; G: ②;  
H: ③。
3. A: ②; B: ③; C: ②; D: ①; E: ④; F: ①; G: ③;  
H: ②; I: ③; J: ①。

4. A: 易破坏的内容先保留的次序; B: 作业控制语言; C: 选择空块链表中第一个不小于申请要求的空块加以切割和分配; D: 进程的调度信息和运行现场; E: 段号和段内地址; F: 当前资源不管如何分配, 系统都不会死锁; G: 在运行过程中优先数可以随时发生变化; H: 时间片轮转分配 CPU; I: 提高资源的利用率和系统的吞吐率; J: 通过中断。

5. A: 12 次; B: 11 次; C: 9 次。

6. A: 11:50 结束; B: 11:20 结束; C: 15:30 结束; D: 13:10 结束; E: 13:40 结束; F: 14:50 结束。

7. 定义 4 个信号量:  $S_1$ 、 $S_2$ 、 $S_3$ 、 $S_4$ , 它们的初值分别为 1、0、1、0。程序:

进程 P	进程 Q	进程 S
for i=1,n do	for i=1,n do	for i=1,n do
begin	begin	begin
P( $S_1$ );	P( $S_2$ );	P( $S_4$ );
PUT Block into K;	P( $S_3$ );	PRINT Block T;
V( $S_2$ );	MOVE Block from K to T;	V( $S_3$ );
end;	V( $S_1$ );	end;
	V( $S_4$ );	
	end;	

8. 文件的最大长度为  $10+512+512^2+512^3+512^4$  页。

9. 最少启动磁盘 6 次, 最多 11 次。

10. 以两个进程都执行同一个信号量 (例如 S) 上的 P 操作为例说明。

令 S 的出值为 1, 用以控制互斥区。显然, 在正常的情况下, 允许一个进程进入。

P(S)的定义:

$S=S-1$ ;

If  $S < 0$  then 调用进程进入相应队列等待。

如果两个进程都调用 P 操作试图进入互斥区, 但 P 操作部互斥执行。当 A 进程调用 P 操作, 核心执行了  $S=S-1$  后转而执行 B 进程调用的 P 操作, 又执行一次  $S=S-1$ 。此时 S 的值成了 -1, 无论进程 A 或 B 都不能进入互斥区。实际上, 两个进程中可以有一个进入。如果互斥执行, 那么先执行 P(S)的进程将被允许进入互斥区, 晚执行的进程将等待, 直到前一个进程退出互斥区、执行 V(S)后才唤醒它进入。

## 第3章 数据库基础知识

### 3.1 内容提要

本章内容主要包括数据库管理技术的发展、数据的描述、数据联系的描述、数据模型和数据库结构、关系代数等、SQL 数据库的数据体系结构、SQL 语言的组成、SQL 语言的使用、数据库的设计过程等。其中关系代数的运算和 SQL 语言的使用是学习的重点。

#### 3.1.1 数据库管理系统的功能和特征

##### 内容要点

(1) 数据库系统的概念：数据库系统 (DataBase System, DBS) 是由数据库、硬件、软件 (数据库管理系统, DataBase Management System, DBMS) 和人员组成, 其中管理的对象是数据。数据是经过组织的比特的集合, 而信息是具有特定释义和意义的

数据。

(2) DBMS 是数据库系统的核心软件, 要在操作系统的支持下工作, 解决如何科学地组织和储存数据, 如何高效地获取和维护数据的系统软件。其主要功能包括: 数据定义功能、数据操纵功能、数据库的运行管理和数据库的建立与维护等。

(3) DBMS 的特征包括: 数据结构化且统一管理、数据与程序独立、具有较强的数据控制功能等。

(4) DBMS 的数据控制功能包括: 数据库的安全性保护、数据的完整性、并发控制、故障恢复等。

#### 3.1.2 数据库管理技术的发展

##### 内容要点

(1) 数据库管理技术的发展阶段: 大致经历了三个阶段, 分别是人工管理阶段、文件系统阶段和数据库阶段。各阶段数据管理的特点是不同的。

(2) 数据库技术的几个基本概念。

① 数据库 (DB): DB 是存储在一起的相关数据的集合, 它能为各种用户共享, 具有最小冗余度, 数据间联系密切, 而又有较高的程序与数据的独立性。

② 数据库管理系统 (DBMS): DBMS 是位于用户与操作系统之间的一层数据管理软件, 为用户或应用程序提供访问 DB 的方法, 包括 DB 的建立、查询、更新及各种数

据控制。

③ 数据库系统 (DBS): DBS 是实现有组织地、动态地存储大量关联数据, 方便多用户访问的计算机软件、硬件和数据资源而组成的系统, 即采用了数据库技术的计算机系统。

④ 数据库技术: 数据库技术是研究数据库的结构、存储、设计、管理和使用的一门软件学科, 它是在操作系统的文件系统基础上发展起来的, 而 DBMS 本身要在操作系统支持下才能工作。

#### 学习难点

本节内容属于数据库系统的基础知识, 涉及一些重要的概念; 考生应重视对这些概念的理解。

#### (1) 数据库管理技术各发展阶段的数据管理特点

① 人工管理阶段数据管理的特点: 数据不保存; 没有专用的软件对数据进行管理; 只有程序概念, 没有文件概念; 数据是面向程序的, 不能重复使用。

② 文件系统阶段的数据管理特点: 数据可长期保存在磁盘上; 由操作系统的文件系统提供存取方法; 文件组织多样化, 但文件之间相互独立, 没有联系; 数据虽然可以重复使用, 但是程序仍然基于特定的物理结构和存取方法; 数据的冗余度大, 数据不具有—致性, 数据联系弱。

③ 数据库阶段的数据管理特点: 使用数据模型, 数据冗余明显减少, 实现了数据共享; 有较高的数据独立性; 数据库系统为用户提供了方便的用户接口; 数据库管理系统提供了四个方面的数据控制功能, 即数据完整性、数据安全性、数据库的并发控制和数据库的恢复

#### (2) 数据库、数据库管理系统和数据库系统三者之间的关系

数据库系统由数据库、数据库管理系统和硬件等组成, 数据库管理系统是数据库系统的核心组成部分。三者之间的关系详见[例 3.2]中的解释。

#### (3) 物理数据独立性和逻辑数据独立性

物理数据逻辑性是指, 当数据的物理结构改变时, 尽量不影响整体逻辑结构、用户逻辑结构以及应用程序; 逻辑数据独立性是指, 在整体逻辑结构改变时, 尽量不影响用户的逻辑结构以及应用程序。

#### (4) 数据库完整性和安全性两个概念的区别

数据库的完整性是指数据库中数据应始终保持正确的状态, 防止不符合语义的错误数据的输入和输出。表达完整性约束的规则有很多形式, 例如属性值的约束, 就是把数据值限制在某个范围内, 并对数据之间联系进行检验。数据库的安全性是指保护数据不被破坏和不被非法窃取。与数据库使用直接有关的安全措施有用户标识和鉴定、存取控制、密码存储和日志监视等方式。



### 3.1.3 数据描述

#### 内容要点

(1) 数据的描述：从事物的特性到计算机里的具体表示，经历了三个数据领域，即现实世界、信息世界和机器世界。现实世界是指客观存在的世界中的事实及其联系；信息世界是现实世界在人们头脑中的反映，是对客观事物及其联系的一种抽象描述；机器世界是在信息世界的基础上的进一步抽象。各个世界的的数据表示方法和使用手段是不同的。

(2) 逻辑数据和物理数据：物理数据描述指数据在存储设备上的存储方式，是实际存放在存储设备上的数据。逻辑数据描述指程序员或用户看到的数据形式，是抽象的概念化数据。在数据系统中，逻辑数据与物理数据之间差别很大。

(3) 数据联系的描述：实体的联系有两类，一类是实体内部的联系，反映在数据上是同一记录内部各字段间的联系；另一类是实体与实体之间的联系，反映在数据上就是记录之间的联系。两个不同实体集的实体间联系有三种情况：一对一联系、一对多联系和多对多联系。

#### 学习难点

(1) 信息世界和机器世界中数据库技术用到的主要术语：现实世界、信息世界和机器世界的关系如图 3-1 所示。在信息世界中，关于数据库技术的主要术语有：实体、实体集、属性、键等；在机器世界，关于数据库技术的主要术语有：字段、记录、文件、键等。信息世界与机器世界中的这些术语既有区别，也有联系。

① 信息世界实体的属性与机器世界的字段相对应：字段是标记实体属性的符号集，是可以命名的最小数据项。字段的命名往往与属性名相同。

② 信息世界的实体与机器世界的记录相对应：在信息世界中，客观存在并且可以相互区别的东西称为实体；在机器世界中，字段的有序集合称为记录。一般地，用一个记录描述一个实体。所以记录又可以定义为能完整描述实体的符号集。

③ 信息世界的实体集与机器世界的文件相对应：在信息世界中，性质相同的同类实体集合称为实体集；在机器世界中，同一类记录的汇集称为文件。文件是描述实体集的，所以它又可以定义为描述一个实体集的所有符号集。

④ 信息世界的键与实体世界的键相对应：在信息世界中，键是能惟一标识实体集

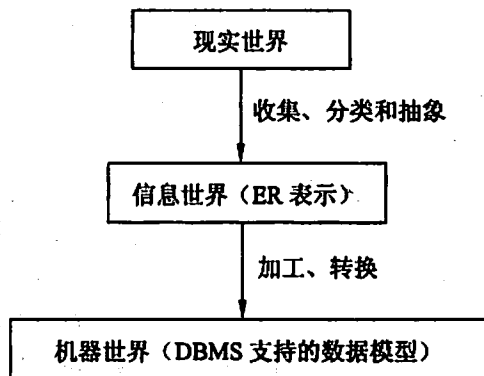


图 3-1 数据管理的三个世界

中每个实体的属性或属性集；在机器世界中，键是指能惟一标识文件中每个记录的字段或字段集。它们的概念是一致的。

(2) 逻辑数据与物理数据的关系：在数据库系统中，逻辑数据与物理数据之间差别很大，用户看到的数据结构和数据，与存储设备中的数据结构和数据可能完全不同。数据库管理软件的功能之一就是要将逻辑数据转换成物理数据，或者把物理数据转换为逻辑数据。

### 3.1.4 数据模型

#### 内容要点

(1) 数据模型的定义：数据库中的数据是有结构的，这种结构反映出事物和事物之间的联系，是按照某种数据模型来组织数据的。数据模型是指表示实体以及实体之间联系的数据库的数据结构。

(2) 数据模型的两种类型：数据模型的种类很多，目前广泛使用的数据模型可分为两种：概念数据模型和基本数据模型。概念数据模型是一种完全独立于任何计算机系统的模型，完全不涉及信息在计算机系统中的应用；基本数据模型则是直接面向数据库的逻辑数据结构，例如层次、网状、关系模型等，也称为结构数据模型，它有严格的形式化定义，以便于在计算机系统中实现。

(3) 基本的数据模型：传统的基本数据模型主要有层次、网状和关系模型等三种。

① 层次模型：层次模型是用树型（层次）结构表示实体类型与实体之间联系的数据模型。

② 网状模型：网状模型用丛结构（网络结构）表示实体类型及实体间的联系。

③ 关系模型：关系模型是用表格结构来表达实体集，用键表示实体间联系。

(4) 在 ER 图中，用矩形框表示实体类型，用菱形框表示实体间的联系类型，用椭圆表示实体或联系的属性，实体间联系用箭头标出并注上联系的种类。

#### 学习难点

(1) 数据模型应包括数据结构、数据操作和完整性约束三个部分：

① 数据结构是指对实体类型和实体之间联系的表达和实现；

② 数据操作是指对数据库的检索与更新（包括插入、删除、修改等）两类操作；

③ 数据的完整性约束给出数据及其联系所具有的制约和依赖规则。

(2) 层次模型、网状模型和关系模型的特点和区别：

① 它们之间的根本区别在于数据之间联系的表示方法不同。在网状和层次模型中联系是用指针实现的，而在关系模型中，基本数据结构是表格，记录之间的联系通过各个关系模式的关键码体现的。

② 层次模型和网状模型的应用程序编制比较复杂，其市场已被关系模型所取代。因为表格简单，用户易懂，用户只需用简单的查询语句就可以对数据进行操作。

③ 层次模型的特征是：有且只有一个结点没有父结点，它就是根结点，其他结点有且只有一个父结点；网状模型的特征是：允许结点有多于一个的父结点，可以有一个以上的结点没有父结点；关系模型的特征是：描述的一致性，不仅用关系描述实体本身，而且也用关系描述实体之间的联系。可直接表示多对多的联系，每个属性不可再分，建立在数学概念基础上，有较强的理论根据。

(3) 概念数据模型中最著名的模型是“实体联系模型”，它直接从现实世界中抽象出实体类型及实体间联系，然后用实体联系图（ER 图）表示数据模型。ER 图的三个要素是：实体、属性和实体之间的联系。数据库设计的第一步就是要使用 ER 图描述数据组织模式，然后进一步转换成任何一种 DBMS 支持的数据模型。

ER 模型建立的一般步骤是：

- ① 确定实体的类型。
- ② 确定实体间联系的类型。
- ③ 根据实体类型和联系类型画出 ER 图。
- ④ 确定实体类型和联系类型的属性。

ER 模型建立方法详见[例 3.11]。

### 3.1.5 数据库系统的结构

#### 内容要点

(1) 数据库的数据体系结构：数据体系结构分成三个级别，分别是内部级、概念级和外部级。这三个结构之间往往差别很大，为实现这三个抽象级别的转换，DBMS 在这三级之间提供了两层映像：外模式/概念模式映像和概念模式/内模式映像。

(2) 概念模式、外模式和内模式：概念模式是数据库中全部数据的整体逻辑结构描述；外模式是用户与数据库系统的接口，是用户用到的那部分数据的描述；内模式是数据库在物理存储方面的描述，包括定义所有的内部数据类型、索引、文件的组织方式，以及数据控制方面的细节。

(3) 外模式/概念模式映像和概念模式/内模式映像：概念模式/内模式映像存在于概念级和内部级之间，用于定义概念模式和内模式间的对应性；外模式/概念模式映像存在于外部级和概念级之间。

(4) 数据库系统的构成：数据库系统是一个复杂的系统，由数据库、硬件支持系统、软件支持系统和数据库管理员（DBA）组成。DBA 的主要职责是：模式定义、存储结构和存取方法的定义、模式和物理组织的修改、对数据库访问的授权、完整性约束的说明等。

(5) 数据库管理系统（DBMS）：DBMS 是指 DBS 中对数据进行管理的软件系统，它是 DBS 的核心成分。它的功能包括：数据库的定义功能、数据库的操纵功能、数据库的保护功能、数据库的维护功能和数据字典。

(6) 数据库系统的全局结构：DBS 的全局结构由数据库用户、数据库管理系统的查

询处理器、数据库管理系统的存储管理器和磁盘存储器中的数据结构等部分组成。

### 学习难点

#### (1) 对数据库体系结构三个级别的理解

① 从某个角度看到的数据特性称为数据视图。外部级最接近用户，是用户能看到的数据特性，用户的数据视图称为“外模型”；

② 概念级是涉及所有用户的数据定义，也就是全局的数据视图，称为“概念模型”；

③ 内部级是最接近于物理存储设备，涉及实际数据的存储方式，物理存储的数据视图称为“内模型”。

这些模型用数据定义语言（DDL）描述后分别得到外模式、概念模式（简称模式）和内模式。子模式是概念模式的子集，可以从概念模式推导出来。各模式的关系如图 3-2 所示。

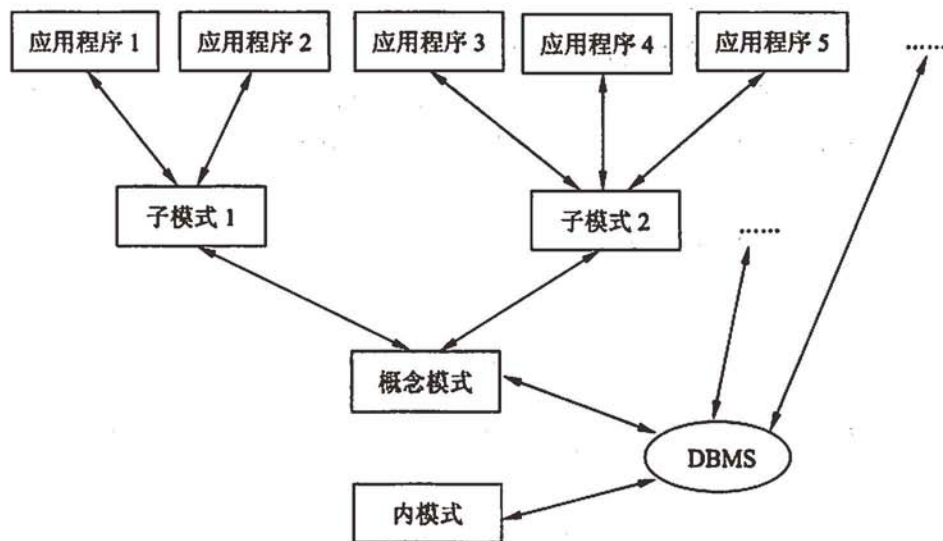


图 3-2 各种模式的关系

#### (2) 两层映像与数据独立性

① 如果数据库的内模式要作修改，即数据库的存储设备和存储方式有所变化，那么概念模式/内模式映像也要作相应的修改，但概念模式很可能仍然保持不变，即达到了物理数据独立性。

② 如果数据库的整体逻辑结构（即概念模式）要作修改，那么外模式/概念模式映像也要作相应的修改，但外模式很可能仍然保持不变，即达到了逻辑数据独立性。

物理数据独立性和逻辑数据独立性是数据库管理系统的重要特性。其目的是当数据的模式或物理模式发生变化时，可以改变相应的映像关系，但保持模式和子模式不变，这样对于应用程序的影响就会很小。



### (3) 数据库的保护功能

DBMS 对数据库的保护主要通过四个方面实现, 即数据库恢复、数据完整性控制、多用户环境下的并发控制和数据安全性控制。

① 数据库恢复: 在数据库系统中, 恢复的基本含义就是恢复数据库本身。也就是说, 在发生某种故障使数据库当前的状态已经不再正确时, 把数据库恢复到已知为正确的某一状态。目前数据库系统中最常用的两种恢复方法是转储和登记日志文件。

② 数据完整性控制: 数据完整性控制是指数据库中数据应始终保持正确的状态, 防止不符合语义的错误数据的输入和输出。表达完整性约束的规则有很多形式, 例如主键约束、引用完整性规则、属性值的约束、断言等。

③ 多用户环境下的并发控制: 在多用户共享系统中, 许多事务可能同时对同一数据进行操作, 称为“并发操作”, 并发操作可能会破坏数据完整性, 或者数据库存储了错误的数据, 或者用户读了不正确的数据(即“脏数据”)。因此 DBMS 中设有并发控制子系统, 负责协调并发事务的执行, 保证数据库的完整性不受破坏。

④ 数据库的安全性: 数据库的安全性是指保护数据不被破坏和不被非法窃取。与数据库使用直接有关的安全措施有用户标识和鉴定、存取控制、审计、密码存储、视图的保护和日志监视等方式。

## 3.1.6 关系模型和关系运算

### 内容要点

(1) 关系模型的基本概念: 关系模型是目前最流行的数据模型, 它用二维表格结构表示实体集, 用关键码表示实体间的联系。关系模型遵循数据库的三级体系结构, 其名称如下: 关系模式、关系子模式和存储模式。关系模型由三部分组成: 数据结构、数据操作、完整性规则。

(2) 关系模型的三类完整性规则: 关系数据库的数据与更新操作必须遵循三类完整性规则: 实体完整性规则、引用完整性规则和用户定义的完整性规则。其中前两个规则是关系模型必须满足的完整性规则, 一般由系统自动支持。用户定义的完整性规则反映某一具体应用所涉及的数据必须满足语义的要求, 具体数据的约束条件由应用环境决定。

(3) 关系代数的五种基本运算: 关系代数是以集合代数为基础发展起来的, 以关系为运算对象的一组高级运算的集合。把关系看成集合, 集合代数中的运算可以引入到关系中来。关系代数的五种基本运算是并、差、笛卡尔积、投影和选择。

① 并: 记为  $R \cup S$ 。由属于  $R$  或属于  $S$  的元组组成, 要求  $R$  和  $S$  具有相同的模式。

② 差: 记为  $R - S$ 。由属于  $R$  但不属于  $S$  的元组组成, 要求  $R$  和  $S$  具有相同的模式。

③ 笛卡尔积: 记为  $R \times S$ 。设关系  $R$  和  $S$  的元数分别是  $r$  和  $s$ , 则  $R \times S$  的元数为  $(r+s)$ , 它的每个元组的前  $r$  个分量来自  $R$  的一个元组, 后  $s$  个分量来自  $S$  的一个元组。若  $R$  有  $m$  个元组,  $S$  有  $n$  个元组, 则  $R \times S$  有  $mn$  个元组。

④ 投影: 记为  $\pi_{i_1, i_2, \dots, i_m}(R)$ 。对关系  $R$  进行垂直分割, 并重新排列的顺序, 再删除重复的元组。显然, 投影后元组的个数可能会减少。

⑤ 选择: 记为  $\sigma_F(R)$ 。根据给定的条件对关系  $R$  作水平分割, 选择符合条件的元组, 元数不变。

(4) 关系代数的四种组合操作:

组合运算指的是它们可以有五种基本运算组合而成。

① 交: 记为  $R \cap S$ , 由既属于  $R$  又属于  $S$  的元组组成, 要求  $R$  和  $S$  具有相同的模式。可以用基本运算实现:  $R \cap S = R - (R - S)$ , 或  $R \cap S = S - (S - R)$ 。

② 联接: 记为  $R \bowtie S$  (有  $\theta$  联接和  $F$  联接两种类型)。在  $R$  和  $S$  的笛卡尔积中挑选第  $i$  个分量和第  $(r+j)$  个分量满足运算的元组 ( $r$  是  $R$  的元数)。它是笛卡尔积和选择两种运算的组合,  $R \bowtie S = \sigma_{i\theta(r+j)}(R \times S)$ 。

③ 自然联接: 记为  $R \bowtie S$ 。它是联接的一个特例, 由笛卡尔积、选择和投影三种运算组合而成, 选择笛卡尔积中那些在两个关系公共属性上的值相等的那些元组, 然后保留一列公共属性。

④ 除法: 记为  $R \div S$ 。假设关系  $R$  和  $S$  的元数分别是  $r$  和  $s$  ( $r > s > 0$ ), 且  $S$  的属性与  $R$  的后  $s$  个属性相同, 则  $R \div S$  是关系  $R$  在前  $r-s$  个属性上投影的一个子集, 该子集与  $S$  的笛卡尔积必须包含在  $R$  中。除法运算由以下基本运算组成:

$$R \div S = \pi_{1, 2, \dots, r-s}(R) - \pi_{1, 2, \dots, r-s}((\pi_{1, 2, \dots, r-s}(R) \times S) - R)$$

### 学习难点

本节学习的重点掌握关系模型的概念和遵循的三类完整性规则, 掌握并能运用关系代数的各种运算, 对于给定的两个关系  $R$  和  $S$ , 应能熟练写出各种运算的结果。

#### (1) 关系模型涉及的主要概念

① 关系: 关系是一个集合, 集合中的成分是元组, 关系的属性的个数称为元数, 各元组的属性个数应当相同。关系虽然与二维表格有类似之处, 但也有区别, 因为关系是一种规范化的数据集合, 在关系模型中, 对关系模型作了一些限制, 如属性不可分解、不可重复, 元组不能重复, 用户不考虑元组的顺序, 但要考虑元组中属性的顺序。

② 关系模式: 关系模式就是记录的类型, 它包括模式名、属性名、值域以及模式的主键。关系模式只是对数据本身特性的描述。

例如, 在学生关系模式  $S(S\#, SNAME, AGE, SEX)$  中,  $S$  是模式名,  $S\#$ 、 $SNAME$ 、 $AGE$  和  $SEX$  是属性名; 每个属性有相应的取值范围, 如  $AGE$  的范围是  $16 \sim 26$ ,  $SEX$  的范围是  $M$  和  $F$  等;  $S\#$  是此模式的键。

③ 键：也称为关键码，是一个属性集合，分为超键、候选键、主键和外键。最常用的是主键，它是正在使用的候选键，指能惟一标识元组的最小属性集。在不加说明的情况下，关系模式的键是指主键。

例如，在学生关系模式  $S(S\#, SNAME, AGE, SEX)$  中，由于属性  $S\#$ （学号）可以惟一确定该模式中的一个元组，所以  $S\#$  是它的键。尽管  $(S\#, SNAME)$  也可以惟一确定一个元组，但是它不是最小集，所以  $(S\#, SNAME)$  不是学生关系模式的键。而在学习关系模式  $SC(S\#, C\#, GRADE)$  中，由于一个学生可能学习多门课程，给定  $S\#$  的值，不能惟一确定一个元组，所以它的键就不是  $S\#$ 。只有  $S\#$  与  $C\#$ （课程名）组合在一起才能惟一确定一个元组，所以学习关系模式的主键是  $(S\#, C\#)$ 。

### (2) 对关系模型的三类完整性规则的理解

关系模型的三类完整性规则是指实体完整性规则、引用完整性规则 and 用户定义的完整性规则。实体完整性规则是指主关键值的任何组成部分不能是空值；引用完整性规则是指不允许引用不存在的实体（即元组）；用户定义的完整性规则针对某一具体数据的约束条件，由应用环境决定。

引用完整性规则可以描述为，如果关系模式  $R_2$  的外关键字  $X$  与关系模式  $R_1$  的主关键字相符，则外关键字的每个值必须在关系模式  $R_1$  中主关键字的值中找到，或者为空值。引用完整性规则的详细解释见[例 3.10]。

### (3) 自然联接运算

根据自然联接的定义，可以得出自然联接运算的运算步骤：

- ① 计算笛卡尔积  $R \times S$ ；
- ② 挑选  $R$  和  $S$  在公共属性上值相等的那些元组；
- ③ 去掉重复的属性，即公共属性只保留一列。

自然联接运算的实例详见[例 3.13]。

这里有两点需要注意：

- 如果两个关系没有公共属性，那么自然连接的结果就是笛卡尔积运算的结果。
- 在自然联接运算中，同名属性一般是外关键字，否则会出现重复数据。

### ④ 关系的除法运算

根据关系除法运算的定义，可以得  $R$  和  $S$  除法的运算步骤：

- ① 计算  $R$  的前  $r-s$  个属性的投影，记为  $R_1$ ；
- ② 计算  $R_1$  与  $S$  的笛卡尔积，记为  $R_2$ ；
- ③ 计算  $R_2 - R$ ，记为  $R_3$ ；
- ④ 计算  $R_3$  的前  $r-s$  个属性的投影，记为  $R_4$ ；
- ⑤ 计算  $R_1 - R_4$ ，此结果就是  $R \div S$  所得的商，记为  $R_5$ 。

$R$  和  $S$  除法的运算实例详见[例 3.14]。

### 3.1.7 关系数据库 SQL 语言简介

#### 内容要点

##### (1) SQL 语言的特点

SQL 语言是集 DDL、DML 和数据控制功能于一体、面向集合的数据库语言。既可以独立使用，又可以嵌入到宿主语言中使用。类似于英语自然语言，简单易学。

##### (2) SQL 语言的组成

① 一个 SQL 数据库是表的汇集，它用一个或多个 SQL 模式定义。

② 一个 SQL 表由行集构成，一行是列的序列，每列对应一个数据项。

③ 一个表或者是一个基本表，或者是一个视图。基本表是实际存储在数据库中的表，而视图是由若干基本表或其他视图构成的表的定义。

④ 一个基本表可以跨一个或多个存储文件，一个存储文件也可以存放一个或多个基本表。用户可以用 SQL 语句对视图和基本表进行查询操作。

##### (3) SQL 的数据定义 (DDL)

SQL 的 DDL 包括以下操作：

① SQL 模式的定义和撤销。

② 基本表的定义、修改（增加或删除属性）和撤销。

③ 视图的定义、更新和操作。

##### (4) SQL 的数据查询语句

包括 SELECT 语句的句型、各子句的格式和使用规定、基本表的联接操作等。

SELECT 语句有六个子句组成，完整的句法是：

```
SELECT      目标表的列名或表达式序列
FROM        基本表或（和）视图序列
[WHERE      条件表达式]
[GROUP BY   列名序列]
[HAVING     组合表达式]
[ORDER BY   列名[序]……]
```

其中前两个子句是比不可少的。

SQL 查询语句中可包括的操作有：条件表达式中的比较操作、属性和基本表的改名操作、集合的并、交、差操作、集合的比较操作等。

当 WHERE 子句、GROUP BY 子句和 HAVING 子句同时出现在一个查询中时，执行的顺序如下：

① 执行 WHERE 子句，从表中选取行。

② 执行 GROUP BY 子句，对选取的行进行分组。



③ 执行聚合函数。

④ 执行 HAVING 子句, 选取满足条件的分组。

(5) SQL 的数据更新语句

包括插入、删除和修改三个语句。

① 插入语句:

```
INSERT INTO R (列名 1, 列名 2, ...)
```

```
VALUES (表达式 1, 表达式 2, ...)
```

功能: 在表 R 中添加新记录, 按列名的顺序, 输入指定字段的数据值。如省略列名列表, 则按照表结构中定义的顺序依次指定每个字段中的值。

② 删除语句:

```
DELETE FROM R
```

```
[WHERE F]
```

功能: 从表 R 中删除满足条件 F 的元组。

③ 修改语句:

```
UPDATE R
```

```
SET 列名=值表达式, [, 列名=值表达式...]
```

```
[WHERE F]
```

功能: 修改表 R 中满足条件 F 的元组的属性值, 修改的值在 SET 语句中给出。如果省略 WHERE 子句, 则该列中的每一行均用同一个值进行更新。

(6) SQL 的访问控制 (授权语句和回收语句)、嵌入式 SQL 等

学习难点

本节学习的重点是 SQL 语句的基本使用方法, 难点是子查询、表的联接查询、集合的比较操作等。为讲述这些学习难点, 假设有以下四个样本表:

学生表: S (学号, 姓名, 性别, 出生日期, 班级)

教师表: T (教师号, 姓名, 性别, 出生日期, 职称, 系别)

课程表: C (课程号, 课程名, 教师号)

成绩表: SC (学号, 课程号, 成绩)

(1) 子查询 (嵌套查询)

子查询可以出现在 WHERE、FROM、HAVING 子句中。下面举例说明。

① 子查询出现在 WHERE 子句中

在学生表中检索与学号为 06 的同学同年同月同日出生的所有学生的学号、姓名和出生日期:

```
SELECT 学号, 姓名, 出生日期
FROM S
WHERE 出生日期= (SELECT 出生日期
                  FROM S
                  WHERE 学号='06')
```

该语句执行的过程是, 先执行子查询: SELECT 出生日期 FROM S WHERE 学号='06', 根据它的结果再执行主查询。

又如, 列出“计算机系”教师所教课程的成绩表:

```
SELECT 课程号, 学号, 成绩
FROM SC
WHERE 课程号 IN (SELECT C.课程号
                 FROM C, T
                 WHERE T.教师号=C.教师号 and T.系别='计算机系')
```

注意上述两例的区别, 前一例的子查询的查询结果只能有一个返回值, 而后一例的子查询的结果可以返回多个值。

### ② 子查询出现在 FROM 子句中

例如, 在成绩表中检索平均成绩超过 80 分的学生学号和平均成绩:

```
SELECT 学号, 平均成绩
FROM (SELECT 学号, AVG (成绩)
      FROM SC
      GROUP BY 学号)
AS RESULT (学号, 平均成绩)
WHERE 平均成绩>80
```

这里 RESULT 是子查询的表名, 它相应的属性名是学号和平均成绩。

### ③ 子查询出现在 HAVING 子句中

例如, 在成绩表中检索平均成绩最高的学生学号:

```
SELECT 学号
FROM SC
GROUP BY 学号
HAVING AVG (成绩) >= ALL (SELECT AVG (成绩)
                          FROM SC
                          GROUP BY 学号)
```

## (2) 表的联接查询

在数据查询中, 经常涉及到提取两个或多个表的数据, 这就需要使用表的联接来实现若干个表数据的联合查询。通常在 WHERE 子句中给出联接条件。

多个表通常存在公共的列，为了区别，在联接条件中通过表名前缀指定连接列。例如，“T.教师号”表示表 T 中的“教师号”列，“C.教师号”表示表 C 中的“教师号”列。例如，检索所有学生的姓名、课程号和成绩：

```
SELECT 姓名, 课程号, 成绩
FROM S, SC
WHERE S.学号=SC.学号
```

在联接查询中，要对两个联接的表做笛卡尔积，因此使用子查询的效率要比使用联接查询的效率高。

### (3) 集合的比较操作

在 WHERE 子句中，可以有四种集合比较操作：集合成员资格比较、集合成员的算术比较、空关系的比较、重复元组的测试。集合成员资格比较 (IN, NOT IN)、集合成员的算术比较 ( $\theta$  SOME,  $\theta$  ALL,  $\theta$  是算术比较运算符) 已在前面的例子中出现过。下面重点讲述空关系比较操作的使用方法。

在子查询中，可以使用 EXSIT，它一般用在 WHERE 子句中，其后紧跟一个子查询，从而构成一个条件，当该子查询至少有一个返回值得时，这个条件成立，否则不成立。NOT EXSIT 与此相反。

例如，检索所有任课教师的姓名和所在系：

```
SELECT 姓名, 系别
FROM T
WHERE EXSIT
    (SELECT *
     FROM C
     WHERE T.教师号=C.教师号)
```

本例执行的过程是，从头到尾扫描表 T，对于每一行，子查询在表 C 中查找是否存在与此行教师号相同的行。如果存在这样的行，EXSIT 子句便返回真 (T.)，条件成立，否则返回假 (F.)，条件不成立。

又如，检索所有未任课的教师的姓名和所在系。

```
SELECT 姓名, 系别
FROM T
WHERE NOT EXSIT
    (SELECT *
     FROM C
     WHERE T.教师号=C.教师号)
```

本例的执行过程与上例基本相同，只是将谓词 EXIST 的结果取反，当子查询找到了

这样的行，则 WHERE 条件为假，否则为真。

### 3.1.8 数据库设计过程

#### 内容要点

(1) 数据库系统生存期：数据库应用系统从开始规划、设计、实现、维护到最后被新的系统所取代而停止使用的整个期间，称为数据库生存期。这个生存期一般可分为七个阶段，即规划、需求分析、概念设计、逻辑设计、物理设计、实现、运行和维护。

(2) 概念设计阶段：概念设计阶段的目标是产生整体数据库概念结构，即概念模式。描述概念结构的工具是 ER 图。利用 ER 方法进行概念设计，可以分成三步进行：首先设计局部的 ER 模式，然后综合成全局的 ER 模式，最后进行全局 ER 模式的优化。

(3) ER 模型向关系模型的转换：一个实体转换为一个关系模式，实体的属性就是关系的属性，实体的标识符就是关系模式的键；把一个联系类型转换为关系模式，所有与该联系相关的实体的键及联系的属性转换成关系的属性，关系模式的键由实体与实体之间的联系所决定。

#### 学习难点

本节学习的主要重点也是难点就是 ER 模型向关系模型的转换。ER 模型的建立方法已在前面介绍过，它的主要成分是实体类型和联系类型。一个实体转换为一个关系模式是比较简单的。难点是将一个联系转换为关系模式，这里要分为三种情况：当联系是 1:1 时，可在任一模式内加入另一个模式的键和联系类型的属性；当联系是 1:M 时，则在 M 端的关系模式中加入 1 端实体类型的键和联系类型的属性；联系是 M:N 时，则联系类型也转换为关系模式，其属性为两端实体类型的键以及联系类型的属性。ER 模型向关系模型的转换实例详见[例 3.11]。

## 3.2 例题分析

【例 3-1】在关系数据模型中，通常可以把 A 称为属性，其值称为属性值，而把 B 称为关系模式。常用的关系运算是关系代数和 C。在关系代数中，对一个关系做投影操作以后，新关系的元组个数 D 原来关系的元组个数。用 E 形式表示实体类型和实体间联系是关系模型的主要特征。

- |          |        |       |       |
|----------|--------|-------|-------|
| A: ①记录   | ②基本表   | ③ 模式  | ④字段   |
| B: ①记录   | ②记录类型  | ③元组   | ④元组集  |
| C: ①集合代数 | ②逻辑演算  | ③关系演算 | ④集合演算 |
| D: ①小于   | ②小于或等于 | ③等于   | ④大于   |
| E: ①指针   | ②链表    | ③关键字  | ④表格   |

正确答案：A: ④ B: ② C: ③ D: ② E: ③



分析:

(1) 在关系数据模型中, 用二维表格结构表示实体类型, 用关键码(关键字)表示实体类型和实体间的联系。字段称为属性, 字段值称为属性值, 记录类型称为关系模型, 记录称为元组, 元组的集合称为关系或实例。

(2) 常用的关系运算是关系代数和关系演算。关系代数是集合代数为基础发展起来的, 以关系为运算对象的一组高级运算的集合。

(3) 投影运算的过程是, 对关系 R 进行垂直分割, 并重新安排列的顺序, 再删除重复的元组。显然, 投影后元组的个数可能会减少。

【例 3-2】 从供选择的答案中选出应填入下列叙述中的\_\_\_\_中。

数据库系统由数据库、A 和硬件等组成, 数据库系统是在 B 的基础上发展起来的。数据库系统由于能够减少数据冗余, 提高数据独立性, 并集中检查 C, 由此获得广泛的应用。数据库管理系统提供给用户的接口是 D, 它具有数据定义、数据操作和数据检查功能, 可独立使用, 也可嵌入宿主语言使用。 E 语言已被国际标准化组织采纳为标准的关系数据库语言。

供选择的答案:

A、B: ① 操作系统    ② 文件系统    ③ 编译系统    ④ 数据库管理系统

C:    ① 数据完整性    ② 数据层次性    ③ 数据操作性    ④ 数据兼容性

D:    ① 数据库语言    ② 过程化语言    ③ 宿主语言    ④ 面向对象语言

E:    ① QUEL    ② SEQUEL    ③ SQL    ④ ALPHA

正确答案: A: ④    B: ②    C: ①    D: ①    E: ③

分析:

(1) 一个完整的数据库系统是由数据库、数据库管理系统和硬件等组成的, 数据库管理系统是为数据库的建立、使用和维护而配置的软件, 是数据库系统的核心组成部分。数据库、数据库管理系统和数据库系统三者之间的关系是:

① 数据库是存储在计算机内、有组织的、可共享的数据的集合。数据库本身不是独立存在的, 它是组成数据库系统的一部分。

② 数据库管理系统属于数据管理软件, 它总是基于某种数据模型, 可分为层次型、网状型、关系型和面向对象型等。

③ 数据库系统是指具有管理和控制数据库功能的计算机系统, 它由四部分组成: 数据库、硬件支持系统、软件支持系统和数据库管理员。其中软件支持系统中的数据库管理系统是数据库系统的核心组成部分。

(2) 在数据库系统出现之前, 数据文件是存储数据的主要形式, 并出现了多种文件组织, 如顺序文件、索引文件、随机文件等。这些类型的数据文件数据冗余度大, 缺乏数据独立性, 数据不能共享和集中管理。在此基础上出现了数据库系统。

(3) 数据库管理系统必须提供统一的数据保护功能, 包括数据的安全性、完整性、

并发控制和恢复。完整性是指保证数据的正确性和有效性。

(4) 数据库管理系统提供给用户的接口是数据库语言。

(5) 结构化查询语言 SQL 是 20 世纪 70 年代初提出的, 现已成为关系数据库的标准语言。

【例 3-3】从供选择的答案中选出应填入下列叙述中的\_\_\_\_中。

在数据库系统中, 与查询有关的是 A; 与规范化方法有关的是 B; 与完整性有关的是 C; 与安全性有关的是 D; 与并发性有关的是 E;

供选择的答案:

A、B: ① 系统目录    ② 进程管理    ③ 页式管理    ④ 索引  
         ⑤ 数据依赖    ⑥ 更新传播

C~E: ① 非过程化    ② 过程化    ③ 数据压缩    ④ 簇聚  
         ⑤ 断言    ⑥ 范式    ⑦ 审计    ⑧ 并发控制

正确答案: A: ④ B: ⑤ C: ⑤ D: ⑦ E: ⑧

分析:

(1) 管理数据库最常用的操作是数据的查询和更新。数据的查询就是按照某种条件在数据库中检索出所需要的数据。

(2) 局部依赖和传递依赖是产生冗余和异常的两个重要原因。低级的范式 (1NF, 2NF) 一般不宜作为数据库的模式, 通常需要将它们变换成 3NF 或更高级的范式, 这种变换过程, 称为“关系的规范化处理”。

(3) 有时一个完整性约束牵涉面较广, 与其他表、完整性约束或规则有关, 这时可用“断言”机制编写完整性规则。

(4) 当对数据的处理极为重要时, 就必须进行审计来追踪检查现有的情况。审计追踪使用的是一个专用文件或数据库, 系统将自动对数据库所有操作记录在上面, 以便重现导致数据库现有状况的一系列事件, 以找出非法存取数据的人。

(5) DBMS 的并发控制子系统负责协调并发事务的执行, 保证数据库的完整性不受破坏, 同时避免用户得到不正确的数据。

【例 3-4】数据模型是用来表示实体和实体之间联系的。网状模型、层次模型和关系模型都是数据库中的基本数据模型。在实体和实体之间联系的表示方法上, 网状模型中采用 A, 层次模型中采用 B, 关系模型中采用 C。在搜索数据时, 层次模型中采用单向搜索法, 网状模型中采用的 D 方法, 关系模型则是通过 E 实现的。

A~C: ① 有向图    ② 连通图    ③ 波特图    ④ 卡诺图  
         ⑤ 结点集    ⑥ 边集    ⑦ 二维表    ⑧ 树

D、E: ① 双向搜索    ② 单向搜索    ③ 循环搜索  
         ④ 可从任一点开始且沿任何路径搜索  
         ⑤ 可从任一结点沿确定的路径搜索

⑥ 可从固定的结点沿任何路径搜索

⑦ 对关系进行运算

正确答案: A: ① B: ⑧ C: ⑦ D: ④ E: ⑦

分析:

(1) 在数据库系统中, 表示实体类型及实体之间联系的模型称为数据模型。传统的基本数据模型主要有层次、网状和关系模型三种。层次结构是用树型结构表示实体类型及实体之间联系; 网状模型是用丛结构(网络结构)表示实体类型及实体之间联系, 网状模型的数据结构是有向图结构; 关系模型是用二维表结构表示实体类型及实体之间联系。

(2) 在网状模型中, 允许有一个以上的结点没有双亲, 结点可以有多个的双亲。因此在网状模型中搜索数据时, 可以从任一结点出发沿任一有向路径搜索。

(3) 在层次模型中, 有且只有一个结点没有双亲, 这个结点称为根结点, 其他结点有且仅有一个双亲。因此在层次模型中搜索数据时, 采用单向搜索方法。

(4) 在关系模型中, 数据是一张二维表, 表中的每一行是一个记录, 而每一个记录由若干个字段组成。因此在关系模型中搜索数据时, 是通过对关系进行运算实现的。

【例 3-5】 从下列关于数据库系统特点的叙述中, 选出 5 条正确的叙述。

(1) 数据库避免了一切数据重复。

(2) 数据库减少了数据冗余。

(3) 各类用户程序均可随意地使用数据库中的各种数据。

(4) 用户程序按所对应的子模式使用数据库中的数据。

(5) 数据库数据可以为经 DBA 认可的各用户所共享。

(6) 数据库系统中如概念模式有改变, 则需要将与其有关的子模式做相应改变, 否则用户程序需改写。

(7) 数据库中的概念模式如有改变, 子模式不必变, 因而用户程序也不必改写。

(8) 数据库系统的存储模式如有改变, 则概念模式应予调整, 否则用户程序会在执行中出错。

(9) 数据库系统的存储模式如有改变, 概念模式无需改动。

(10) 数据一致性是指数据库中的数据类型一致。

正确答案: (2), (4), (5), (7), (9)

分析:

(1) 数据库的特征有: 实现数据共享, 减少数据冗余度, 保持数据的一致性, 数据的独立性, 安全保密性, 并发控制, 故障恢复等。数据库不能避免一切重复, 有时为了提高效率还会增加重复数据。因此本题说法是错误的。

(2) 减少冗余是数据库的特征之一。因此本题说法是正确的。

(3) 数据库系统提供了安全性措施, 对用户的数据给予合理的保护, 对于获得上机权的, 还要进一步根据用户权限控制操作的范围。防止未经许可的用户对有关数据的访



问和修改。因此本题说法是错误的。

(4) 关系模型遵循数据库的三级体系结构, 其名称如下: 关系模式、关系子模式和存储模式。关系模式是数据库的概念模式, 定义为关系模式的集合; 存储模式是对数据库物理存储结构的描述。关系子模式是用户所用到的那部分数据的描述, 是用户与数据库的接口, 也称为用户对数据库的视图。用户对数据库的操作, 实际上就是对子模式记录进行操作。因此本题说法是正确的。

(5) 虽然数据共享是数据库的特征之一, 但为了数据的安全性, 用户共享数据必须取得数据库管理员 (DBA) 的授权。因此本题说法是正确的。

(6) 在数据库的三级模式结构中, 概念模式是关系模式的集合, 关系模式仅仅是对数据本身特性的描述。子模式定义了各子模式与模式之间的映射关系。当整个系统要求改变模式时, 可以改变映射关系而保持子模式不变, 也不会影响以子模式方式使用数据库的用户程序。这种用户数据独立于全局的逻辑数据的特性叫做逻辑数据独立性。因此本题说法是错误的。

(7) 根据上题的分析, 可知本题的说法是正确的。

(8) 概念模式相对于存储模式是独立的, 概念模式的改变不会影响存储模式, 同样, 存储模式的改变也不会影响概念模式。当为了某种需要改变存储模式时, 可以同时改变两者之间的映射而保持模式和子模式的不变, 这种全局的逻辑数据独立于物理数据的特性叫做物理数据独立性。逻辑数据独立性和物理数据独立性是数据库管理系统的重要特性。因此本题说法是错误的。

(9) 根据上题的分析, 存储模式的改变不会影响概念模式, 因此, 本题说法是正确的。

(10) 数据的一致性是指表示同一数据的多个副本之间没有矛盾, 完全一致, 并不是指数据库中的数据类型的一致。因此本题说法是错误的。

【例 3-6】域表达式  $\{ab \mid R(ab) \wedge R(ba)\}$  转换为等价的关系代数表达式, 所列出的式子中, A 是不正确的。在 SQL 中, 集合成员资格的比较操作“元组 NOT IN (集合)”中的“NOT IN”与 B 操作符等价。SQL 中涉及属性 AGE 是否是空值的比较操作, 写法 C 是错误的。类似于“工资在 800~5000 之间”这种约束, 是属于 DBS 的 D 功能。

关系 R 和 S 的自然连接运算一般只用于 R 和 S 有公共 E 的情况。

A: ①  $\pi_{1,2}(\sigma_{1=4 \wedge 2=3}(R \times R))$                       ②  $\pi_{1,2}(\sigma_{1=4}(R \bowtie_{2=3} R))$

③  $\pi_{1,2}(\sigma_{1=4}(R \bowtie_{2=1} R))$                       ④  $R \cap \pi_{2,1}(R)$

B: ①  $\diamond$  SOME                      ② =SOME                      ③  $\diamond$  ALL                      ④ =ALL

C: ① AGE IS NULL                      ② NOT (AGE IS NULL)

③ AGE=NULL                      ④ AGE IS NOT NULL



- D: ① 完整性                      ② 并发控制                      ③ 安全性                      ④ 恢复  
 E: ① 元组                          ② 属性                          ③ 关键字                      ④ 关系模式  
 正确答案: A: ②    B: ③ C: ③    D: ①    E: ②

分析:

先来看问题 A, 它涉及域表达式与关系代数表达式转换的问题。域表达式  $\{ab|R(ab) \wedge R(ba)\}$  的意义是取二元关系 R 中有对称关系的二元组的集合, 即  $(a, b) \in R, (b, a) \in R$ 。提供的答案中只有第二个是不正确的, 它的自然连接的条件不是第一个 R 的第二个元素与第二个 R 的第一个元素相等。

SQL 语句中的集合比较有四种: 集合成员资格比较、集合成员算术比较、空关系测试和重复元组的测试。问题 B 是关于集合成员资格的比较, 它有两种形式:

<集合 1> IN <集合 2>  
 <集合 1> NOT IN <集合 2>

其中“IN”与算术比较中的“=SOME”等价, “NOT IN”与算术比较中的“<>ALL”等价。

SQL 中允许属性值为空值, 用关键字 NULL 表示空值。例如, 测试“年龄为空值”的条件可用“AGE IS NULL”来表示, 测试“年龄为非空值”的条件可用“AGE IS NOT NULL”来表示。

属性值的约束属于数据库完整性范畴。表达完整性的约束的规则有很多形式, 例如主键约束、引用完整性规则、属性值的约束、断言等。

【例 3-7】关系型数据库语言 (SQL) 目前得到了越来越广泛的使用。SQL 基本的使用方式有两种。它可以单独使用, 称为 A; 也可以在高级语言编写的应用程序中使用, 称为 B, 这时相应的高级语言称为 C。从 SQL 数据库的体系结构角度来看, 用户可以用 SQL 语言的语句, 对 D 和 E 进行查询操作, 用户可把它们都看做为关系 (表格), 但是 E 是一个或几个 D 导出的表, 它本身不独立存储在数据库中。

- A、B: ① 宿主语言                      ② 嵌入式语言                      ③ 数据定义语言  
           ④ 交互式语言                      ⑤ 数据操纵语言  
 C: ① 元语言                          ② 目标语言                      ③ 源语言                      ④ 宿主语言  
 D、E: ① 游标                          ② 视图                          ③ 库文件  
           ④ 基本表                          ⑤ 存储文件

正确答案: A: ④    B: ②    C: ④    D: ④    E: ②

分析:

(1) SQL 语言的基本使用方式有两种: 一种是单独使用, 即供用户在交互环境下使用, 也称为交互语言; 另一种使用方式是嵌入式 SQL, 将 SQL 语句直接嵌入某种高级语言中使用, 如嵌入到 C、Pascal 等语言中, 这时相应的高级语言称为宿主语言。

(2) SQL 语言的操作对象和结果都是关系。这种关系既可以是一个基本表, 也可以

是一个或几个导出表,称为视图。视图是一种“虚表”,它本身并不独立存储在数据库中,基本表则是一张实表,每个基本表对应一个存储区域。

【例 3.8】关系数据库语言 SQL 是一种 A 语言,使用方便。

若要在基本表 S 中增加一列 CN (课程名),可用 B;

若要撤销数据库中已存在的表 S,可用 C。

设关系数据库中一个表 S 的结构为 S (SN, CN, grade),其中 SN 为学生名, CN 为课程名,两者均为字符型; grade 为成绩,是数值型,取值范围为 0~100。

若要把“王二的化学成绩 80 分”插入到 S 中,则可用 D;

若需要更正王二的成绩为 85 分,则可用 E。

A: ① 高级算法      ② 过程性      ③ 汇编      ④ 说明性

B: ① ADD TABLE S (CNCHAR (8))

② ADD TABLE S ALTER (CNCHAR (8))

③ ALTER TABLE ADD (CNCHAR (8))

④ ALTER TABLE S (ADD CNCHAR (8))

C: ① DEL TABLE S      ② DEL S

③ DROP TABLE S      ④ DROP S

D: ① ADD INTO S

VALUES ('王二', '化学', '80')

② INSERT INTO S

VALUES ('王二', '化学', '80')

③ ADD INTO S

VALUES ('王二', '化学', 80)

④ INSERT INTO S

VALUES ('王二', '化学', 80)

E: ① UPDATE S

SET grade=85

WHERE SN='王二'AND CN='化学'

② UPDATE S

SET grade='85'

WHERE SN='王二'AND CN='化学'

③ UPDATE grade=85

WHERE SN='王二'AND CN='化学'

④ UPDATE grade='85'

WHERE SN='王二'AND CN='化学'

正确答案: A: ④ B: ③ C: ③ D: ④ E: ①

**分析:**

(1) SQL 是关系型数据库的典型语言。它是一种非过程化(说明性)语言。所谓说明性是指在使用时,只需说明“做什么”,而不必给出“如何做”的详细过程。

(2) 在表中增加一列,属于基本表的扩充,语句格式为:

ALTER TABLE 表名

ADD 列名 类型

因此在基本表 S 中增加一列 CN(课程名),可用:

ALTER TABLE S

ADD (CN CHAR(8))

(3) 基本表的撤销语句格式为:

DROP TABLE 表名

因此要撤销数据库中已存在的表 S,可用:

DROP TABLE S

(4) 插入语句的格式为:

INSERT INTO R(列名 1, 列名 2, ...)

VALUES(表达式 1, 表达式 2, ...)

功能:在表 R 中添加新记录,按列名的顺序,输入指定字段的数据值。如省略列名列表,则按照表结构中定义的顺序依次指定每个字段中的值。因此若把“王二的化学成绩 80 分”插入到 S 中,可用:

INSERT INTO S

VALUES('王二','化学',80)

(5) 修改语句的格式为:

UPDATE R

SET 列名=值表达式, [列名=值表达式...]

[WHERE F]

功能:修改表 R 中满足条件 F 的元组的属性值,修改的值在 SET 语句中给出。如果省略 WHERE 子句,则该列中的每一行均用同一个值进行更新。因此如需要更正王二的化学成绩为 85 分,可用:

UPDATE S

SET grade=85

WHERE SN='王二'AND CN='化学'

**【例 3-9】**为了保证数据库中数据的安全可靠有正确有效,数据库管理系统(DBMS)提供数据库恢复、并发控制、数据完整性保护与数据安全保护等功能。数据库在运行过程中由于软硬件故障可能造成数据被破坏,数据库恢复就是在尽可能短的时间内,把数据库恢复到故障发生前的状态。具体的实现方法有多种,如:定期将数据库作 A:在

进行事务处理时,对数据更新(插入、删除、修改)的全部有关内容写入 B;在系统正常运行时,按一定的时间间隔,设立 C,把内存缓冲区内容还未写入磁盘中去的相关状态记录到 C;当发生故障时,根据现场数据内容、D 的故障前映像和 E 来恢复系统的状态。

A~E: ①库文件            ②日志文件            ③检查点文件    ④后备文件  
         ⑤主文件            ⑥源程序            ⑦流文件        ⑧作业

正确答案: A: ④ B: ② C: ③ D: ② E: ③

分析:

(1) 在 DBMS 运行过程中,有可能出现各种各样的故障。在数据库系统中,恢复的基本含义就是恢复数据库本身。也就是说,在发生某种故障使数据库当前的状态已经不再正确时,把数据库恢复到已知为正确的某一状态。

(2) 恢复的基本方法有:

- ① 周期性地对整个数据库进行复制,或转储到磁带一类的存储介质中。
- ② 建立“日志”数据库。日志文件是用来记录对数据库每一次更新活动的文件。
- ③ 在系统正常运转时,按一定的时间间隔设立检查点文件,把内存缓冲区中还未写入到磁盘中的有关状态记录到检查点文件中。
- ④ 如果数据库系统出现了故障,根据现场数据内容、日志文件的故障前映像和检查点文件来恢复系统的状态。

【例 3-10】举例说明引用完整性规则。

解:

假设有如图 3-3 所示的两个关系模式:成绩关系模式和课程关系模式。在成绩关系模式中,学号是关键字,课程号是外关键字;在课程关系模式中,课程号是关键字,根据引用完整性规则的定义,R2 是成绩关系模式,R1 是课程关系模式。

对于该例,引用完整性规则的含义是,成绩关系模式中课程号的值或者为空值,或者在课程关系模式中的课程号中能够找到。满足这个条件是必须的,如果不满足,假如成绩关系模式中某个元组课程号的属性值是 k16,由于在课程关系模式中课程号的值中找不到,则该课程号显然是不正确的,这样会造成数据的不一致性。

学号	姓名	课程号	成绩
106	刘国华	k6	82
112	李军	k18	91
116	苏朋	k6	86
208	张丽燕	k12	99
122	李晓华	k12	76

课程号	课程名	任课教师
k6	微机原理	袁立
k8	程序设计语言	王国明
k10	操作系统	赵利军
k12	高等数学	王爱素
k18	会计学	李蔓

图 3-3 成绩与课程关系模式



**【例 3-11】** 用 ER 图描述图书信息管理的数据库模型，并将 ER 模型转换为关系模型。  
解：

图书信息管理的数据库模型是：每个借书人有姓名、借书证号和单位属性，每个借书人可以借 5 本书，每本图书有书名、书号、数量和位置属性。

首先确定实体的类型，这里有两个实体集，即借书人和图书；其次，由于同一种书可以有几个借阅人同时借阅，一个借阅人可以同时借阅多本书，所以它们之间是多对多的关系，联系的类型是借阅；第三，画出对应的 ER 图，如图 3-4 所示；最后确定实体类型和联系类型的属性，实体借书人的属性有姓名、证号和单位；实体图书的属性有书名、书号、数量和位置；联系类型借阅的属性有借书日期和还书日期。

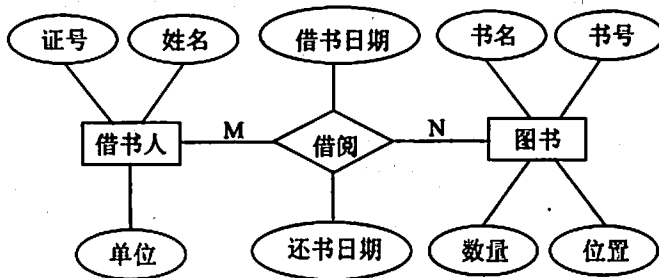


图 3-4 图书借阅管理 ER 图

ER 模型向关系模型的转换。由于任何人可借多种书，任何一种书可为多个人所借，所以这两个联系是  $M:N$  的实体，借书证的证号具有惟一性。可转换成如下的关系模式：

借书人模式：借书人（证号，姓名，单位）

图书模式：（书号，书名，数量，位置）

借阅模式：借阅（证号，书号，借阅日期，还书日期）

这里，借书人模式的键是证号，它能惟一确定借书人的姓名和单位；图书模式的键是书号，它能惟一确定一种书的书名、数量和位置；借阅模式是通过两个实体之间的联系形成的，它的属性包括两端实体类型的键（借书证号，书名）和联系类型的属性（借阅日期，还书日期）。

**【例 3-12】** 设有如下实体：

学生：学号，单位，姓名，性别，年龄，选修课程名

课程：编号，课程名，开课单位，任课教师号

教师：教师号，姓名，性别，职称，讲授课程编号

单位：单位名称，电话，教师号，教师名

上述实体存在如下联系：

(1) 一个学生可以选修多门课程，一门课程可为多个学生选修；

(2) 一个教师可讲授多门课程，一门课程可为多个教师讲授；

(3) 一个单位可有多个教师，一个教师只能属于一个单位。

试完成以下工作：

(1) 分别设计学生选课和教师任课两个局部信息的结构 ER 图。

(2) 将上述设计完成的 ER 图合并成一个全局 ER 图。

(3) 将该全局 ER 图转换为等价的关系模型表示的数据库逻辑结构。

解：

(1) 学生选课局部 ER 图如图 3-5 所示，教师任课局部 ER 图如图 3-6 所示。

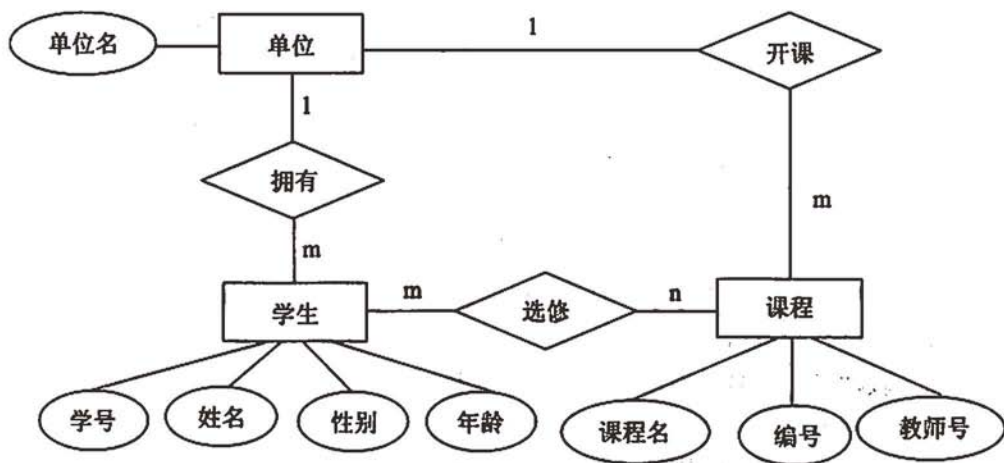


图 3-5 学生选课局部 ER 图

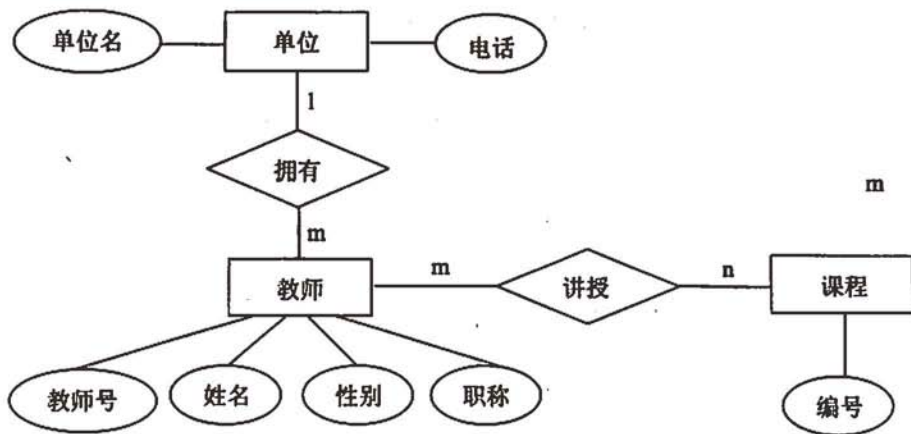


图 3-6 教师任课局部 ER 图

(2) 合并的全局 ER 图如图 3-7 所示。图中各实体的属性是：

单位：单位名，电话

学生：学号，姓名，性别，年龄

教师：教师号，姓名，性别，职称

课程：编号，课程名

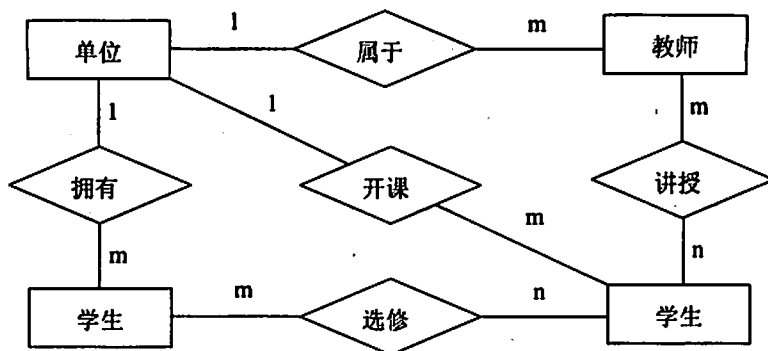


图 3-7 合并的全局 ER 图

(3) 该全局 ER 图转换为等价的关系模型表示的数据库逻辑结构如下：

单位（单位名，电话）

教师（教师名，姓名，性别，职称，单位名）

课程（课程编号，课程名，单位名）

学生（学号，姓名，性别，年龄，单位名）

讲授（教师号，课程编号）

选修（学号，课程号）

【例 3-13】 图 3-8 是关系 R 和关系 S，求 R 和 S 的自然联接。

R				S		
P	Q	T	Y	T	Y	B
2	b	c	d	c	d	m
9	a	e	f	c	d	n
2	b	e	f	d	f	n
9	a	d	e			
7	g	e	f			
7	g	c	d			

图 3-8 关系 R 和关系 S

解：图 3-9 是 R 和 S 自然联接的运算过程。

P	Q	T	Y	T	Y	B
2	b	c	d	c	d	m
9	a	e	f	c	d	m
2	b	e	f	c	d	m
9	a	d	e	c	d	m
7	g	e	f	c	d	m
7	g	c	d	c	d	m
2	b	c	d	c	d	n
9	a	e	f	c	d	n
2	b	e	f	c	d	n
9	a	d	e	c	d	n
7	g	e	f	c	d	n
7	g	c	d	c	d	n
2	b	c	d	d	f	n
9	a	e	f	d	f	n
2	b	e	f	d	f	n
9	a	d	e	d	f	n
7	g	e	f	d	f	n
7	g	c	d	d	f	n

(a)  $R \times S$ 

P	Q	R.T	R.Y	S.T	S.Y	B
2	b	c	d	c	d	m
7	g	c	d	c	d	m
2	b	c	d	c	d	n
7	g	c	d	c	d	n

(b) 挑选在公共属性上值相等的元组

P	Q	T	Y	B
2	b	c	d	m
7	g	c	d	m
2	b	c	d	n
7	g	c	d	n

(c) 去掉重复的属性

图 3-9 R 和 S 自然联接的运算过程

【例 3-14】 图 3-10 是关系 R 和关系 S，求 R 和 S 的除法运算结果。

R			
P	Q	T	Y
2	b	c	d
9	a	e	f
2	b	e	f
9	a	d	e
7	g	e	f
7	g	c	d

S	
T	Y
c	d
e	f

图 3-10 关系 R 和关系 S

解：图 3-11 是 R 和 S 除法的运算过程。



R1		R2				R3				R4		R5	
P	Q	P	Q	T	Y	P	Q	T	Y	P	Q	P	Q
2	b	2	b	c	d	9	a	c	d	9	a	2	b
9	a	9	a	c	d							7	g
7	g	7	g	c	d								
		2	b	e	f								
		9	a	e	f								
		7	g	e	f								

图 3-11 R ÷ S 的运算过程

### 3.3 思考练习题及答案

#### 思考练习题

##### 一、单项选择题

- 下列\_\_\_\_不是文件系统阶段数据管理的缺点。
  - 数据冗余性大
  - 数据不能保存
  - 数据不一致
  - 数据联系弱
- 客观存在的各种报表、图表和查询格式等原始数据属于\_\_\_\_。
  - 机器世界
  - 信息世界
  - 现实世界
  - 模型世界
- \_\_\_\_是位于用户和操作系统之间的一层数据管理软件。
  - 数据库
  - 数据库技术
  - 数据库系统
  - 数据库管理系统
- 对数据库进行保护,防止未经授权的或不合法的使用造成的数据泄露、更改破坏。这是指数据的\_\_\_\_。
  - 安全性
  - 恢复
  - 并发控制
  - 完整性
- 数据库的\_\_\_\_是指数据的正确性和相容性。
  - 安全性
  - 恢复
  - 并发控制
  - 完整性
- 数据库管理系统通常提供授权功能来控制不同用户访问数据的权限,这主要是为了实现数据的\_\_\_\_。
  - 一致性
  - 可靠性
  - 安全性
  - 完整性
- 用于数据库恢复的重要文件是\_\_\_\_。
  - 日志文件
  - 备份文件
  - 恢复文件
  - 备注文件
- 处理并发控制的主要方法是采用\_\_\_\_。
  - 事务技术
  - 封锁技术
  - 保护技术
  - 密码技术

9. 如果不对并发操作加以控制, 则会产生\_\_\_\_。
- A 数据被非法窃取    B 不一致分析问题    C 系统死锁    D 事务混乱
10. 在数据库技术中, 对于\_\_\_\_的定义称为“授权”。
- A 用户代号    B 用户标识    C 存取权限    D 读写控制
11. 数据库系统与文件系统的主要区别是\_\_\_\_。
- A 数据库系统复杂, 而文件系统简单
- B 文件系统不能解决数据冗余和数据独立性问题, 而数据库系统可以解决
- C 文件系统只能管理程序文件, 而数据库系统能够管理各种类型的文件
- D 文件系统管理的数据量较少, 而数据库系统可以管理庞大的数据量
12. 下列关于数据库系统的正确叙述是\_\_\_\_。
- A 数据库系统减少了数据的冗余
- B 数据库系统避免了一切数据冗余
- C 数据库系统中的数据的一致性是指数据类型的一致性
- D 数据库系统比文件系统能够管理更多的数据
13. 数据库的特点之一是数据的共享, 这里的数据共享是指\_\_\_\_。
- A 同一个应用中的多个程序共享一个数据集合
- B 多个用户、同一种语言共享数据
- C 多个用户共享一个数据文件
- D 多种应用、多种语言、多个用户相互覆盖地使用数据集合
14. 以下叙述中, 正确的是\_\_\_\_。
- A 在数据库系统中, 数据独立性指数据之间的相互独立, 互不依赖
- B 对数据库的查找、增添、删除和修改等操作都需要由数据库管理员进行完整性定义和安全性授权, 有数据库系统具体执行
- C 在数据库系统中, 数据的完整性是指数据的正确性和有效性
- D “授权”是数据库系统中采用的完整性措施之一
15. 以下叙述中, 不正确的是\_\_\_\_。
- A SQL 语言的视图定义和视图操作功能不支持逻辑数据的独立性
- B SQL 语言中不提供显式地使用索引的功能, 支持了物理数据的独立性
- C 引用完整性规则是指不允许引用不存在的实体
- D 实体完整性规则是指主关键字的任何组成部分都不可以是空值
16. 数据库管理系统能实现对数据库中的数据查询、插入、修改和删除等操作, 这种功能称为\_\_\_\_。
- A 数据定义功能    B 数据管理功能
- C 数据操纵功能    D 数据控制功能
17. 在数据库的三级模式结构中, 描述数据库中全体数据的全局逻辑结构和特征

是\_\_\_\_\_。

A 外模式

B 内模式

C 存储模式

D 模式

18. 应用数据库的主要目的是为了\_\_\_\_\_。

A 解决数据的保密问题

B 解决数据的共享问题

C 解决数据量大的问题

D 解决数据的完整性问题

19. 关系数据模型\_\_\_\_\_。

A 只能表示实体间的 1:1 联系

B 只能表示实体间的 1:n 联系

C 只能表示实体间的 m:n 联系

D 可以表示实体间的上述三种联系

20. 子模式是指\_\_\_\_\_。

A 模式的副本

B 多个模式的集合

C 模式的逻辑子集

D 数据库的部分元组

21. 在关系模型中, 主键是指\_\_\_\_\_。

A 能惟一标识元组的一组属性集

B 用户正在使用的候选键

C 模型的第一个属性或第二个属性

D 以上说法都不正确

22. 在关系代数运算中, 五种基本运算是指\_\_\_\_\_。

A 并、差、笛卡尔积、投影、选择

B 并、差、交、投影、选择

C 并、差、联接、投影、选择

D 联接、除法、笛卡尔积、投影、选择

23. 一般情况下, 当对关系 R 和 S 进行自然联接时, 要求 R 和 S 含有一个或多个共有的\_\_\_\_\_。

A 子模式

B 记录

C 属性

D 元组

24. 如下图所示, 两个关系 R1 和 R2, 它们进行\_\_\_\_\_运算后得到 R3。

A 交

B 联接

C 笛卡尔积

D 自然联接

R1		
A	B	C
a	1	x
c	2	y
c	1	y

R2		
B	D	E
1	m	i
2	n	j
5	m	k

R3				
A	B	C	D	E
a	1	x	m	i
c	1	y	m	i
c	2	y	n	j

25. 设有关系 R(A, B, C) 和 S(B, C, D), 下列各关系代数表达式不成立的是\_\_\_\_\_。

A

B

C

D

26. 设有属性 A, B, C, D, 以下表示中不是关系的是\_\_\_\_\_。

A R (A)

B R (A, B, C, D)

C R (A×B)

D R (A, B)

27. SQL 语言是\_\_\_\_\_的语言, 易学习。

A 非过程化

B 过程化

C 格式化

D 导航式

28. 下列 SQL 语句中, 修改表结构的是\_\_\_\_\_。

A UPDATE

B ALTER

C INSERT

D CREAT

29. 假定学生关系是 S (学号, 姓名, 性别, 年龄), 课程关系是 C (课程号, 课程名, 教师), 学生选课关系是 SC (学号, 课程号, 成绩)。要查找选修“数据库原理”课程的“女”学生姓名, 将涉及关系\_\_\_\_\_。

A SC, C

B SC

C S, C

D S, C, SC

30. 假定有三个关系, 学生关系 S、课程关系 C 和学生选课关系 SC, 他们的结构如下:

S (S#, SN, SEX, AGE, DEPT)

C (C#, CN)

SC (S#, C#, GRADE)

其中, S#为学生号, SN 为姓名, SEX 为性别, AGE 为年龄, DEPT 为系别, C#为课程号, CN 为课程名, GRADE 为成绩。检索所有比“李军”年龄大的学生的姓名、年龄和性别, 正确的 SQL 语句是\_\_\_\_\_。

A SELECT SN, AGE, SEX

FROM S

WHERE AGE> (SELECT AGE

FROM S

WHERE SN= “李军”)

B SELECT SN, AGE, SEX

FROM S

WHERE AGE> SN= “李军” )

C SELECT SN, AGE, SEX

FROM S

WHERE AGE> (SELECT AGE

WHERE SN= “李军” )

D SELECT SN, AGE, SEX

FROM S

WHERE AGE>李军.AGE

31. 各关系模式同上题, 检索选修课程“C2”的学生中成绩最高的学生的学号, 正



确的 SQL 语句是\_\_\_\_\_。

- A SELECT S#  
FROM SC  
WHERE C#="C2" AND GRADE IN  
(SELECT GRADE  
FROM SC  
WHERE C#="C2" )
- B SELECT S#  
FROM SC  
WHERE C#="C2" AND GRADE >=  
(SELECT GRADE  
FROM SC  
WHERE C#="C2" )
- C SELECT S#  
FROM SC  
WHERE C#="C2" AND GRADE >=ALL  
(SELECT GRADE  
FROM SC  
WHERE C#="C2" )
- D SELECT S#  
FROM SC  
WHERE C#="C2" AND GRADE NOT IN  
(SELECT GRADE  
FROM SC  
WHERE C#="C2" )

32. 各关系模式同第 30 题, 检索学生姓名及其所选修课程的课程号和成绩, 正确的 SQL 语句是\_\_\_\_\_。

- A SELECT S.SN, SC.C#, SC.GRADE  
FROM S  
WHERE S.S#=SC.S#
- B SELECT S.SN, SC.C#, SC.GRADE  
FROM SC  
WHERE S.S#=SC.S#
- C SELECT S.SN, SC.C#, SC.GRADE  
FROM S

D SELECT S.SN, SC.C#, SC.GRADE  
FROM S, SC  
WHERE S.S#=SC.S#

33. 各关系模式同 30 题, 检索选修四门以上课程的学生总成绩 (不统计不及格的课程), 并要求按总成绩的降序排列, 正确的 SQL 语句是\_\_\_\_\_。

A SELECT S#, SUM (GRADE)  
FROM SC  
WHERE GRADE>=60  
ORDER BY 2 DESC  
GROUP BY S#  
HAVING COUNT (\*) >=4

B SELECT S#, SUM (GRADE)  
FROM SC  
WHERE GRADE>=60  
HAVING COUNT (\*) >=4  
GROUP BY S#  
ORDER BY 2 DESC

C SELECT S#, SUM (GRADE)  
FROM SC  
WHERE GRADE>=60  
GROUP BY S#  
HAVING COUNT (\*) >=4  
ORDER BY 2 DESC

D SELECT S#, SUM (GRADE)  
FROM SC  
WHERE GRADE>=60  
GROUP BY S#  
ORDER BY 2 DESC  
HAVING COUNT (\*) >=4

34. 数据库系统的生存期是指\_\_\_\_\_。

- A 从数据定义开始到完成数据库系统设计为止的期间
- B 从开始规划、设计、实现、维护到最后被新的系统取代而停止使用的整个期间
- C 从概念设计到逻辑设计, 再到物理设计
- D 数据库系统正常运转的整个过程

35. 数据库概念模式产生于数据库系统设计的\_\_\_\_阶段。  
A 规划                      B 需求分析                      C 概念设计                      D 逻辑设计
36. 数据库的逻辑设计阶段就是\_\_\_\_。  
A 把 ER 图转换为关系模式的过程  
B 定义数据库各个属性的过程  
C 设计数据库存储结构的过程  
D 建立实际的数据库结构的过程

## 二、填空题

1. 数据库的保护包括\_\_\_\_、\_\_\_\_、\_\_\_\_和\_\_\_\_四个方面。
2. 人工管理阶段的数据管理中, 只有\_\_\_\_概念, 没有\_\_\_\_概念。(文件/程序)
3. 数据库的数据模型包含三个部分, 其中, \_\_\_\_是指对实体类型和实体之间联系的表达和实现; \_\_\_\_是指对数据库的检索和更新两大类操作; \_\_\_\_给出数据及其联系所具有的制约和依赖规则。
4. 数据库的数据体系结构分成三个级别, 分别是\_\_\_\_、\_\_\_\_和\_\_\_\_。
5. 概念模式是数据库中全部数据的整体\_\_\_\_的描述; 外模式是\_\_\_\_与数据库系统的接口; 内模式是数据库在\_\_\_\_方面的描述。
6. 数据库系统的核心组成部分是\_\_\_\_。
7. ER 数据模型一般在数据库设计的\_\_\_\_阶段使用。
8. 数据库语言包括\_\_\_\_和\_\_\_\_两大部分, 前者负责描述和定义数据库的各种特性, 后者用于说明对数据进行的各种操作。
9. 当数据的物理存储改变了, 应用程序不变, 而由 DBMS 处理这种改变, 这是指数据的\_\_\_\_。
10. 目前流行的基本数据类型有三类, 它们是\_\_\_\_、\_\_\_\_和\_\_\_\_。
11. 关系模型遵循数据库的三级体系结构, 其名称如下: \_\_\_\_、\_\_\_\_和\_\_\_\_。
12. 关系中每一个属性都有一个取值范围, 这个取值范围称为属性的\_\_\_\_。
13. 关系数据库采用\_\_\_\_作为数据的组织方式。
14. 关系数据库基于数学上的两类运算是\_\_\_\_和\_\_\_\_。
15. 关系代数中, 从两个关系中找出相同的元组的运算, 称为\_\_\_\_运算。
16. 联接操作是\_\_\_\_和\_\_\_\_操作的组合。
17. 关系代数是\_\_\_\_代数为基础发展起来的, 它是以\_\_\_\_为运算对象的一组高级运算的集合。
18. 设有如下页图所示的关系 S、SC 和 C, 试用关系代数表达式表示下列查询语句:  
(1) 检索“程军”老师所授课程的课程号和课程名, 语句是\_\_\_\_。  
(2) 检索年龄大于 21 的男生的学号和姓名, 语句是\_\_\_\_。

S				SC			SC		
S#	SNAME	AGE	SEX	C#	CNAME	TEACHER	S#	C#	GRADE
1	李强	23	男	k1	C++语言	王华	1	k1	83
2	刘丽	22	女	k5	数据库	程军	2	k1	85
5	张友	22	男	k8	编译原理	程军	5	k1	92
							2	k5	90
							5	k5	84
							5	k8	80

- (3) 检索至少选修“程军”老师所授全部课程的学生姓名, 语句是\_\_\_\_\_。
- (4) 检索“李强”同学不学课程的课程号, 语句是\_\_\_\_\_。
- (5) 检索至少选修两门课程的学生的学号, 语句是\_\_\_\_\_。
- (6) 检索全部学生都选修的课程的课程号和课程名, 语句是\_\_\_\_\_。
- (7) 检索选修课程包括“程军”老师所授课程之一的学生的学号, 语句是\_\_\_\_\_。
- (8) 检索选修课程号为 k1 和 k5 的学生的学号, 语句是\_\_\_\_\_。
- (9) 检索选修全部课程的学生的姓名, 语句是\_\_\_\_\_。
- (10) 检索选修课程包含学号为 2 的学生所修课程的学生的学号, 语句是\_\_\_\_\_。
- (11) 检索选修课程名为“C++语言”的学生的学号和姓名, 语句是\_\_\_\_\_。

19. 设有关系 R (BH, XM, XB, DWH), S (DWH, DWM) 和 T (BH, XM, XB, DWH), 则:

- (1) 实现  $R \cup T$  的 SQL 语句是\_\_\_\_\_。
- (2) 实现  $\sigma_{DWH=100}(R)$  的 SQL 语句是\_\_\_\_\_。
- (3) 实现  $\pi_{XM, XB}(R)$  的 SQL 语句是\_\_\_\_\_。
- (4) 实现  $\pi_{XM, DWH}(\sigma_{XB='女'}(R))$  的 SQL 语句是\_\_\_\_\_。
- (5) 实现  $R \bowtie S$  的 SQL 语句是\_\_\_\_\_。
- (6) 实现  $\pi_{XM, XB, DWM}(\sigma_{XB='男'}(R \bowtie S))$  的 SQL 语句是\_\_\_\_\_。

20. 设有如下的关系 R (NO, NAME, SEX, AGE, CLASS), 主关键字是 NO, 其中 NO 为学号, NAME 为姓名, SEX 为性别, AGE 为年龄, CLASS 为班号。写出实现下列功能的 SQL 语句:

- (1) 插入一个记录, (25, “李明”, “男”, 21, “95031”): \_\_\_\_\_。
- (2) 插入“95031”班学号为 30、姓名为“郑和”的学生记录: \_\_\_\_\_。
- (3) 将学号为 10 的学生姓名改成“王华”: \_\_\_\_\_。
- (4) 将所有“95101”班号改成“95091”: \_\_\_\_\_。
- (5) 删除学号为 20 的学生记录: \_\_\_\_\_。
- (6) 删除姓“王”的学生记录: \_\_\_\_\_。



## 思考练习题答案

### 一、单项选择题

1. B 2. A 3. D 4. C 5. A 6. C 7. D 8. B 9. A 10. B  
 11. B 12. A 13. D 14. C 15. A 16. C 17. D 18. B 19. D 20. C  
 21. B 22. A 23. C 24. D 25. B 26. C 27. A 28. B 29. D 30. A  
 31. C 32. D 33. C 34. B 35. C 36. A

### 二、填空题

1. 数据库的恢复, 完整性控制, 并发控制, 安全控制
2. 程序, 文件
3. 数据结构, 数据操作, 数据的完整性约束
4. 内部级, 概念级, 外部级
5. 逻辑结构, 用户, 物理存储
6. 数据库管理系统
7. 概念设计
8. 数据描述语言, 数据操纵语言
9. 物理独立性
10. 关系模型, 层次模型, 网状模型
11. 关系模式, 关系子模式, 存储模式
12. 值域
13. 关系模型
14. 关系代数, 关系演算
15. 交
16. 笛卡尔积, 选择
17. 集合, 关系
18. (1)  $\pi_{S\#, CNAME}(\sigma_{TEACHER='WY'}(C))$   
 (2)  $\pi_{S\#, CNAME}(\sigma_{AGE>21 \wedge SEX='男'}(C))$   
 (3)  $\pi_{SNAME}(S \bowtie (\pi_{S\#, C\#}(SC) \div \pi_{C\#}(\sigma_{TEACHER='WY'}(C))))$   
 (4)  $\pi_{C\#}(C) - \pi_{C\#}(\sigma_{SNAME='李强'}(S) \bowtie SC)$   
 (5)  $\pi_{S\#}(\sigma_{[1]=[4] \wedge [2] \neq [5]}(SC \times SC))$   
 (6)  $\pi_{C\#, CNAME}(C \bowtie \pi_{S\#, C\#}(SC) \div \pi_{S\#}(S))$   
 (7)  $\pi_{S\#}(SC \bowtie \pi_{C\#}(\sigma_{TEACHER='WY'}(C)))$   
 (8)  $\pi_{S\#, C\#}(SC) \div \pi_{C\#}(\sigma_{C\#='K1' \vee C\#='K5'}(C))$   
 (9)  $\pi_{SNAME}(S \bowtie (\pi_{S\#, C\#}(SC) \div \pi_{C\#}(C)))$   
 (10)  $\pi_{S\#, C\#}(SC) \div \pi_{C\#}(\sigma_{C\#=2}(SC))$

(11)  $\pi_{S\#, SNAME}(S \bowtie \pi_{S\#}(SC \bowtie (\sigma_{CNAME='C' \rightarrow 王}(C))))$

19. (1) SELECT \* FROM R

UNION

SELECT \* FROM T

(2) SELECT \* FROM R

WHERE DWH='100'

(3) SELECT XM, XB FROM R

(4) SELECT XM, DWH FROM R

WHERE XB='女'

(5) SELECT R.BH, R.XM, R.DWH, S.DWM

FROM R

WHERE R.DWH=S.DWH

(6) SELECT R.XM, R.XB, S.DWM

FROM R, S

WHERE R.DWH=S.DWH AND R.XB='男'

20. (1) INSERT INTO R

VALUES (25, "李明", "男", 21, "95031")

(2) INSERT INTO R (NO, NAME, CLASS)

VALUES (30, "郑和", "男", "95031")

(3) UPDATE R

SET NAME="王华"

WHERE NO=10

(4) UPDATE R

SET CLASS="95091"

WHERE CLASS="95031"

(5) DELETE FROM R

WHERE NO=20

(6) DELETE FROM R

WHERE NAME LIKE "王%"

## 第4章 多媒体基础知识

根据对过去试题的分析,考试的内容主要集中在多媒体的计算方面,其次是多媒体的存储技术,而多媒体网络应用和多媒体编辑语言的题目不多。为了回答这些试题,笔者推荐使用清华大学出版社 2002 年 9 月出版的林福宗编著的《多媒体技术基础》第 2 版教材,可作为复习参考。

本章所用术语取自“全国科学技术名词审定委员会”推荐的技术术语,如用“视像(video)”而不用“视频”,用声音(audio)而不用“音频”等,以尽可能避免产生物理概念上的错误。

对照《全国计算机技术与软件专业资格(水平)考试/程序员考试大纲》和《全国计算机技术与软件专业资格(水平)考试/软件设计师考试大纲》,发现在多媒体技术基础部分没有多大差别。此外,通过对程序员试题和设计师试题的分析,发现试题的难度差别也不是很大,有些程序员试题甚至比设计师试题还难以回答。考虑到上述两种情况,程序员和设计师都需要掌握本章介绍的内容。

本章中的部分例题、练习题及其分析选自公开出版的书籍和过去的试题,例题和练习题后面的小标题是笔者加的。这些试题和笔者对部分试题的点评,对读者理解和掌握多媒体基础知识应该是很有帮助的。由于多种原因,没有列出参考文献的名称,在此谨向作者表示感谢。

### 4.1 内容提要

#### 4.1.1 多媒体的概念

多媒体(multimedia)是组合两种或两种以上媒体的人机交互式信息传播媒体。使用的媒体包括文字、图形、图像、声音、动画和电视图像等。多媒体是超媒体(hypermedia)系统中的一个子集,而超媒体系统是使用超链接(hyperlink)构成的全球信息系统,全球信息系统是因特网上使用 TCP/IP 协议和 UDP/IP 协议的应用系统。二维的多媒体网页使用超文本标记语言(HTML)、可扩展标记语言(XML)等语言编写,三维的多媒体网页使用虚拟现实建模语言(VRML)等语言编写。目前许多多媒体作品使用光盘发行,以后将更多地使用网络发行。

多媒体技术基础分成四个主要组成部分:(1)多媒体的表示和计算,涵盖文字、声音、图像和数字电视媒体的基本知识和编码方法;(2)多媒体的存储技术,涵盖 CD

和 DVD 存储器的存储原理和多媒体在存储器中的存放格式; (3) 多媒体网络应用, 涵盖多媒体网络应用的特点、因特网、多目标广播和多媒体通信系统的基础知识; (4) 多媒体内容编辑语言, 涵盖超文本标记语言 (HTML) 和可扩展标记语言 (XML) 的基础知识。

### 4.1.2 多媒体计算技术

#### 1. 数据压缩

数据压缩是减少数据所占用的存储空间或者减少传输数据所占用的带宽的方法。数据压缩可分为无损压缩 (Lossless Compression) 和有损压缩 (Lossy Compression) 两种类型。

无损压缩是指使用压缩后的数据进行重构 (或叫做还原、解压缩), 重构后的数据与原来的数据完全相同的数据压缩技术; 无损压缩用于要求重构的数据与原始数据完全一致的场合, 如磁盘文件的压缩。根据目前的技术水平和文件本身的冗余程度, 无损压缩算法通常可把普通文件大小压缩到原来的 25~50%。常用的无损压缩算法有哈夫曼算法和 LZW 算法。

有损压缩是指使用压缩后的数据进行重构, 重构后的数据与原来的数据有所不同, 但不影响人对原始资料表达的信息造成误解。有损压缩适用于重构数据不一定非要和原始数据完全相同的场合。例如, 图像和声音的压缩就可以采用有损压缩, 因为其中包含的数据往往多于我们的视觉系统和听觉系统所能接收的信息, 丢掉一些数据而不至于对声音或图像所表达的意思产生误解, 但可大大提高压缩比。

#### 2. 声音技术

声音是携带信息的极其重要的媒体, 是多媒体技术研究中的重要内容。声音按照频率范围划分, 通常分成下面几种类型:

- ① 高保声音信号 (High-Fidelity Audio): 10~20 000 Hz
- ② 声音 (Audio): 20~20 000 Hz
- ③ 语音 (Speech): 300~3000/3400 Hz
- ④ 亚音信号/次音信号 (Subsonic): <20 Hz
- ⑤ 超声 (Ultrasonic): >20 000 Hz

声音技术主要包括: 声音数字化、声音数据压缩、语音识别和文语转换 (text to speech, TTS) 等; 要重点掌握的是声音数字化中的采样和量化这两个最基本的概念, 其次是经常遇到的声音文件存储格式; 要搞清楚的概念是语音识别和文语转换。

声音 (Audio) 是 Sound 的同义词, 语音 (Speech) 是声音中的一部分。由于它们的主要用途不同以及它们的频率范围不同, 因此处理方法也不同。例如在压缩技术中, 前者主要使用波形编码、音源编码和混合编码技术, 后者主要使用心理声学模型。

要记住两种声音的采样参数: (1) 声音 (Audio), 采样频率为 44.1 kHz, 样本精度



为 16 位,许多压缩算法和设备的数据率都是从这两个参数导出的。(2) 话音 (Speech), 采样频率为 8 kHz, 样本精度为 8 位,许多编码器和通信设备的数据率都是从这两参数导出的。

### 3. 图像技术

图像是多媒体中携带信息的极其重要的媒体,有人发表过统计资料,认为人们获取信息的 70% 来自视觉系统,因此图像一直是一个热门研究领域。要进入这个领域,例如要进入图像搜索引擎的研究或技术开发工作,就需要有宽范围的基础知识,包括颜色空间、图像的表达方法、图像文件格式、图像数据压缩等。

颜色(color)是人的视觉系统对可见光的感知结果,感知到的颜色由波长在 380 nm~780 nm 之间的光波决定。通常认为,人的视网膜上有对红、绿、蓝颜色敏感程度不同的三种锥体细胞,还有一种在光功率极低的条件下才起作用的杆状体细胞。红、绿、蓝三种锥体细胞对不同波长的光的感知程度不同,对不同亮度的感知程度也不同。这些锥体细胞采样得到的信号通过大脑产生不同颜色的感觉,这些感觉由国际照明委员会(CIE)作了定义,用色调、饱和度和明度三个特性区分颜色,它们是颜色固有的截然不同的特性。

图像数字化之后的数据量非常大,在因特网上传输时很费时间,在盘上存储时很占“地盘”,因此数十年来,许多科技工作者一直在孜孜不倦地寻找更有效的图像压缩方法,用比较少的数据表达原始图像。图像数据压缩主要根据下面两个基本事实来实现的。一个是图像数据中有许多重复的数据,使用数学方法来表示就可以减小数据量;另一个是人的眼睛对图像细节和颜色的辨认有一个极限,把超过极限的部分去掉,这也就达到压缩数据的目的。科技工作者长期努力的一个结果是 JPEG 压缩标准的诞生。

### 4. 数字电视

电视是当代最有影响力的信息传播工具!电视是 20 世纪 20 年代的伟大发明,在 50 年代开发电视技术时,用任何一种数字技术来传输和再现真实世界的图像和声音都是极其困难的,因此电视技术一直沿着模拟信号处理技术的方向发展,直到 70 年代才开始开发数字电视。由于数字技术具有许多优越性,而且数字技术发展到了足以使模拟电视向数字电视过渡的水平,电视和计算机才开始融合在一起。

电视包含视像和声音两个部分,要掌握的基本知识包括:

(1) 彩色电视制式(NTSC 制、PAL 制和 SECAM 制)的主要特性,包括每帧的扫描线数和帧速率等。

(2) 电视图像信号数字化,尤其要理解 ITU-R BT.601 标准(原为 CCITT 601)中的重要参数的含义,以及各种子采样格式。

(3) 电视系统的颜色空间转换。从技术上角度区分,颜色空间可考虑分成如下三类:

① RGB 型颜色空间/计算机图形颜色空间:主要用于电视机和计算机的颜色显示系统。例如,RGB, HSI, HSL 和 HSV 等颜色空间。② XYZ 型颜色空间/CIE 颜色空间:这是

由国际照明委员会定义的颜色空间,通常作为国际性的颜色空间标准,用作颜色的基本度量方法。国际照明委员会定义的颜色空间是与设备无关的颜色表示法,在科学计算中得到广泛应用。对不能直接相互转换的两个颜色空间,可利用这类颜色空间作为过渡性的颜色空间,例如, CIE 1931 XYZ,  $L^*a^*b$ ,  $L^*u^*v$  和 LCH 等颜色空间就可作为过渡性的转换空间。③ YUV 型颜色空间/电视系统颜色空间: 由广播电视需求的推动而开发的颜色空间,主要目的是通过压缩色度信息以有效地播送彩色电视图像。例如, YUV, YIQ, ITU—R BT.601 Y'CbCr, ITU—R BT.709 Y'CbCr 和 SMPTE—240M Y'PbPr 等颜色空间。

(4) MPEG—Video 和 MPEG—Audio 的压缩原理等。MPEG 标准一直是许多科研机构 and 大学的科研热点,也是工业界产品开发的热点。MPEG 标准阐明了声音和电视图像的编码和解码过程,严格规定了声音和图像数据编码后组成比特数据流的句法,提供了解码器的测试方法等,但没有对所有内容都作严格规定,尤其是对压缩和解压缩的算法,这样既保证了解码器能对符合 MPEG 标准的语音数据和电视图像数据进行正确解码,又给 MPEG 标准的具体实现留有很大余地。人们可以不断改进编码和解码算法,提高声音和电视图像的质量以及编码效率。

了解和掌握这些基础知识,无论对理解和开发视像软硬件产品还是对进入多媒体检索这个研究热点都是非常必要的。

### 4.1.3 多媒体存储技术

光盘如何记录“0”和“1”,如何提高单位面积上的记录密度是计算机工业中的一个非常重要的技术研究和开发课题。在半个世纪中,科学家和工程技术人员开发了许多记录技术,从电子管到半导体存储器,从磁记录到光记录都取得了辉煌的成就。光记录是 20 世纪 70 年代的重大发明,是 80 年代世界上的重大技术开发项目,是 90 年代得到广泛应用的技术。

CD 在多媒体的发展过程中立下了不朽的功勋,CD 和 DVD 光盘是发展中国家发行多媒体文件的重要媒体。要了解和掌握的基础知识包括:

(1) 工作原理: CD (Compact Disc) 盘是利用激光和机械方法在盘上压制凹坑来记录二进制数字“0”和“1”的数据存储器。它的记录原理既不同于磁盘,也不同于磁光盘 (Magneto—Optical Disk) 和相变光盘 (Phase Change Disk),它利用凹坑的边缘来记录“1”,利用凹坑和非凹坑的平坦部分记录“0”,其平坦部分的长度代表“0”的个数。使用激光读出时,凹坑反射的光的强度比从非凹坑反射的光的强度弱,由此来区分“1”和“0”。这些“1”和“0”是二进制数据经过 8—14 调制 (Eight-to-Fourteen Modulation, EFM) 之后的位,叫做通道位。使用这种方法比直接用凹坑和非凹坑代表原始二进制数据的“0”和“1”更有效。

(2) 激光唱盘: 用于存储数字声音数据的光存储媒体。光盘上存放的声音是高保真

立体声，声音的采样频率为 44.1 kHz，样本精度为 16 位，一张光盘上声音的连续播放时间可达 74 分钟。光盘是一张无磁性的、光亮而带有保护塑料衬垫的金属盘，直径为 120 mm（4.75 英寸），通常写成 5 英寸，厚度为 1.2 mm，重 14 g~18 g，使用激光扫描机构读出盘上的数据。

(3) DVD（Digital Versatile Disc / Digital Video Disc）盘：用来存储影视、声音和其他数字数据的光盘，1995 年开始开发。它的工作原理与 CD 盘相同，盘片的直径都是 12 mm（4.75 英寸），但存储容量比 CD 盘大得多。一张标准的单层单面数字视盘能存储 4.7 GB 数据，一张标准的双层单面数字视盘其容量可达 8.5 GB，一张双面双层数字视盘的容量可达 17 GB。提高 DVD 容量的主要技术是采用波长比较短的激光，以减小螺旋光道之间的距离和凹坑的长度，采用双面记录和扩大记录面积等，见图 4-1 和表 4-1 中。

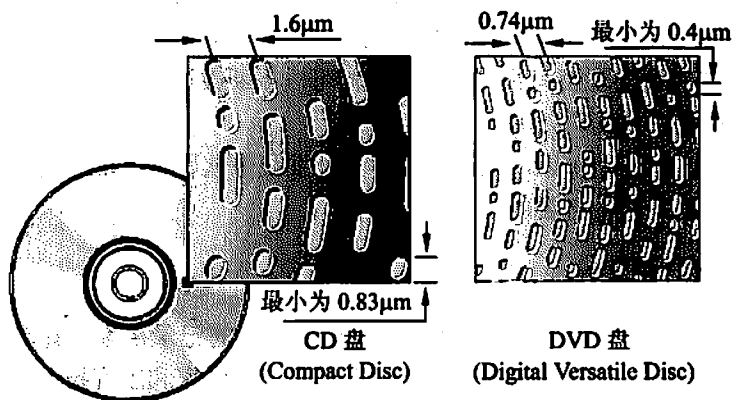


图 4-1 DVD 与 CD 盘的比较

表 4-1 DVD 技术摘要

	DVD 盘	CD 盘	容量增益
盘片直径(mm)	120	120	
盘片厚度(mm/面)	0.6	1.2	
减小激光波长(nm)	635/650	780	4.486 =
加大 N.A.(数值孔径)	0.6	0.45	(1.6×0.83)/
减小光道间距(μm)	0.74	1.6	(0.74×0.40)
减小最小凹凸坑长度(μm)	0.4	0.83	
减小纠错码的长度	RSPC	CIRC	
修改信号调制方式	8-16	8-14 加 3	1.0625 = 17/16
加大盘片表面的利用率	86.6 平方厘米	86 平方厘米	1.019 = 86.6/86
减小每个扇区字节数	2048/2060 字节/扇区	2048/2352 字节/扇区	1.142 = 2352/2060



(4) 存储格式: CD 和 DVD 格式都包含逻辑格式和物理格式。逻辑格式实际上是文件格式的同义词, 它规定如何把文件组织到光盘上以及指定文件在光盘上的物理位置, 包括文件的目录结构、文件大小以及所需盘片数目等事项; 物理格式则规定数据如何放在光盘上, 这些数据包括物理扇区的地址、数据的类型、数据块的大小、错误检测和校正码等。CD 格式详细记载在标准文件中, 如图 4-2 所示。这些标准文件包括红皮书、黄皮书、ISO 9660、绿皮书、橙皮书和白皮书等, 而且还在不断推出。CD 的标准文件是用彩色封面包装的, 所以又称为彩书标准。DVD 格式与 CD 格式有许多相同之处, 了解了 CD 格式, DVD 格式就会迎刃而解。

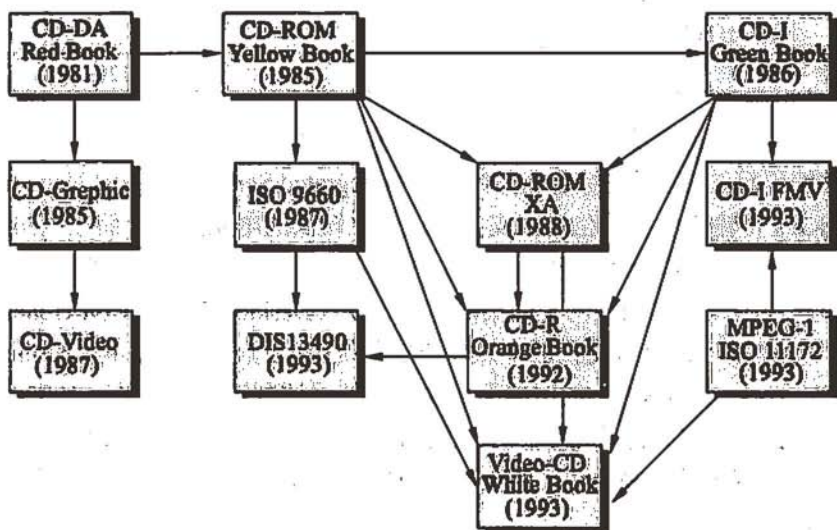


图 4-2 CD 标准系列

#### 4.1.4 多媒体网络应用

多媒体网络技术 (Multimedia Networking) 是目前网络应用开发的热门技术之一。因特网上已经开发了很多应用, 归纳起来大致可分成两类, 一类是以文本为主的数据通信, 包括文件传输、电子邮件、远程登录、网络新闻和 Web 等, 另一类是以声音和电视图像为主的通信。通常把任何一种声音通信和图像通信的网络应用称为多媒体网络应用 (Multimedia Networking Application)。网络上的多媒体通信应用和数据通信应用有比较大的差别, 前者要求在客户端播放声音和图像时要流畅, 声音和图像要同步, 因此对网络的时延和带宽要求很高。而数据通信应用则把可靠性放在第一位, 对网络的时延和带宽的要求不那么苛刻。

多媒体网络应用涉及到的基础知识面很广, 主要包括多媒体计算技术、网络技术和



多媒体通信标准等。其中, 网络技术在网络课程中有专门的介绍。涉及的多媒体通信标准见表 4-2。

表 4-2 三个主要的多媒体通信系列标准

	H.320	H.323(V1/V2)	H.324
发布时间	1990	1996/1998	1996
应用范围	窄带 ISDN	带宽无保证信息包交换网络	PSTN
图像编码	H.261, H.263	H.261, H.263, H.264	H.261, H.263, H.264
声音编码	G.711, G.722, G.728	G.711, G.722, G.728G.723.1, G.729	G.723.1
多路复合控制	H.221, H.230/H.242	H.225.0, H.245	H.223, H.245
多点	H.231, H.243	H.323	
数据	T.120	T.120	T.120

20 世纪 90 年代初开发的电视会议标准是 H.320, 它定义通信的建立、数字电视图像和声音压缩编码算法, 运行在综合业务数字网(Integrated Services Digital Network, ISDN)上, 在 56 Kb/s 传输率的通信信道上支持帧速率比较低的电视图像, 而在 1.544 Mb/s 传输率的信道(即 T1 信道)上可以传输公用中分辨格式(Common Intermediate Format, CIF)的满帧速率电视图像; 在局域网上的桌面电视会议(Desktop Video Conferencing)采用 H.323 标准, 这是基于信息包交换的多媒体通信系统; 在公众交换电话网(Public Switched Telephone Network, PSTN)上的网桌面电视会议使用调制解调器, 采用 H.324 标准。因特网上的电视会议目前大部分都趋向于采用 H.323 标准和正在开发的 IP 电话标准, 使用 IP 协议提供局域网上的电视会议, 而全球的因特网电视会议目前还不能保证实时电视会议的服务质量。

公用中分辨格式(CIF)在多媒体网络应用中也是经常用到的知识, 它是用在可视通信中按图像大小定义的标准电视图像格式, 见表 4-3。最初的 CIF 格式称 FCIF(Full CIF)格式。表中的数据位速率是没有经过压缩的位速率。

表 4-3 公用中分辨格式

CIF 格式	图像大小	每秒 30 帧时的位速率(Mbps)
SQCIF (sub quarter CIF)	128×96	4.4
QCIF (quarter CIF)	176×144	9.1
CIF (Full CIF, FCIF)	352×288	36.5
4 CIF (4×CIF)	704×576	146.0
16 CIF (16×CIF)	1048×1152	583

其他常见的术语包括:

H.261 (1993 年 3 月)——H.261 视像编码标准: 国际电信联盟 (ITU) 推荐的电视图像压缩标准, 主要用来支持可视电话和电视会议, 并于 1992 年开始应用于综合业务

数字网络 (ISDN)。该标准采用帧内压缩和帧间压缩技术, 可使用硬件或软件执行。电视图像数据压缩后的数据速率为  $p \times 64$  Kbps, 其中  $p$  的变动范围为  $1 \sim 30$ , 这取决于所使用的 ISDN 通道数。H.261 支持 CIF 和 QCIF 的分辨率。

H.263 (1998 年 2 月)——H.263 视像编码标准: 国际电信联盟 (ITU) 推荐的用于低位速率通信的电视图像编码标准, 目的是改善在调制解调器上传输的图像质量。H.263 支持 CIF, QCIF, SQCIF, 4CIF 和 16CIF 的图像分辨率。

H.264 (2002 年 9 月)——H.264 视像编码标准: ITU-T 推荐的视像编码工业标准。在相同质量下, 比现有 MPEG-Video 的压缩效率提高  $2 \sim 3$  倍。ITU-T 视像编码专家组 (Video Coding Experts Group, VCEG) 从 1995 年开始开发低位速率的可视通信, 其草案标准称为 H.26L, 2001 年 MPEG 专家组认识到 H.263L 的潜力, 并成立由 MPEG 和 VCEG 专家组成的联合视像组 (Joint Video Team, JVT), 其结果产生了名称不同内容一致的两个国际标准, 常写成 ITU-T Rec. H.264 /MPEG-4 AVC (Advanced Video Coding) 或 ISO/IEC MPEG4 Part 10。

G.711——G.711 语音编码标准: 国际电信联盟 (ITU) 推荐的数据率为 64 Kbps 的脉冲编码调制 (PCM) 语音编码标准, 使用 A-率或  $\mu$  率压缩和扩展编码方法。

G.721——G.721 语音编码标准: 国际电信联盟 (ITU) 推荐的数据率为 32 Kbps 的 ADPCM 语音编码标准。

G.722——G.722 语音编码标准: 国际电信联盟 (ITU) 推荐的数据速率为 64 Kbps、声音带宽为 7 KHz 的语音编码标准。

G.722.1——G.722.1 语音编码标准: 国际电信联盟 (ITU) 推荐的数据速率为 24 Kbps、声音带宽为 7 KHz 和数据速率为 32 Kbps、声音带宽为 14 KHz 的语音编码标准。

G.723.1——G.723.1 语音编码标准: 国际电信联盟 (ITU) 推荐的 5.3 Kbps 和 6.3 Kbps 双数据速率声音编码器标准。该标准用于甚低位速率 (very low bit rate) 的多媒体服务, 它是 H.324 标准中的一个成员。6.3 Kbps 速率的声音质量较高些, 5.3 Kbps 速率的声音质量满足通信要求, 并可为系统设计师提供附加灵活性。这两个速率可在任何一帧的边界处进行切换。一帧的时间为 30 ms, 对数据编码时要附加 7.5 ms 的前向预测时间, 因此算法的总延迟时间为 37.5 ms。声音数据采用线性预测时域合成—分析编码技术 (Linear Predictive Analysis—By—Synthesis Coding), 高数据率编码器的激励信号是多脉冲最大似然量化 (MP—MLQ) 信号, 低数据率编码器的激励信号是代数编码激励线性预测 (ACELP) 信号。

G.728——G.728 语音编码标准: 国际电信联盟 (ITU) 推荐的声音数据速率为 16 Kbps、声音带宽为 3.4 KHz 的声音编码标准, 采用码激励线性预测算法 (CELP)。

G.729——G.729 语音编码标准: 国际电信联盟 (ITU) 推荐的声音数据速率为 8 Kbps、声音带宽为 3.4 KHz 的语音编码标准, 采用码激励线性预测算法 (CELP)。

### 4.1.5 多媒体内容编辑语言

文档要便于访问、易于传输、有效检索和灵活多样,这就需要使文档不受软件和硬件的约束,而通用标记语言就是具有这些特性的语言。用来说明文档结构的第一个正式标记语言是由IBM公司于20世纪60年代创造的,称为通用标记语言(Generalized Markup Language, GML),用于内部文档的标准化。其后扩展成标准通用标记语言(Standard General Markup Language, SGML),成为工业上用来表达信息的标准,并于1986年成为国际标准化组织(ISO)的正式标准(SGML ISO 8879—1986)。在这部分中,要掌握如下两个最基本的概念。

(1) 超文本标记语言(Hypertext Markup Language, HTML): 用来创建超文本文档所用的标记语言,它是20世纪90年代从SGML中挑选的一个子集。SGML使用标签来标志文档中的文本或图形、图像元素,并告诉Web浏览器该如何向用户显示这些元素,以及如何响应用户的行为。例如,当用户通过按键或鼠标单击某个链接时该如何响应。HTML 2.0由因特网工程特别工作组(IETF)制定,它包含了1995年所有浏览器的通用特性,并且它是广泛应用于万维网的第一个HTML版本。HTML未来版本的开发将由万维网协会来进行。HTML 3.2集成了许多1996年初所实现的特性。大多数的Web浏览器,特别是Netscape Navigator和Internet Explorer所识别的HTML标记都超过了现有的标准。现在已经开发了HTML 4.x版本,集成的标签更多,功能更丰富。尽管HTML语言有这样那样的问题和局限性,但人们一直在使用和改进它。

(2) 可扩展标记语言(Extensible Markup Language, XML): 万维网协会(World Wide Web Consortium, W3C)推出的开放标准,用于定义Web网页上的文档元素和商业文档,它克服了HTML不能由Web开发人员自己定义标签等不足的地方。XML使用与HTML类似的标签结构,与HTML之间的差别主要是:HTML定义如何显示文档元素,而XML定义包含什么文档元素;HTML使用预先定义的标签,而XML允许网页开发人员定义自己的标签。

目前,使用最广泛的Web标准是HTML和XML。这些标准将为日益增长的多媒体网络应用、多媒体移动通信和多媒体电子出版业等产生深刻的影响。

## 4.2 例题分析

### 4.2.1 多媒体的概念

#### 【例4-1】多媒体媒体数据的特性

多媒体技术中,表达信息的媒体数据具有一些特定的性质。下述关于媒体数据性质的描述中,不正确的是\_\_\_\_\_。



- A. 有格式的数据才能表达信息
- B. 不同的媒体所表达的信息量不同
- C. 媒体之间的关系也代表着信息
- D. 任何媒体都可以直接进行相互转换

分析:

这是一道试题,下面是“2004 试题分析与解答”的全文:

从信息表达的角度来看,媒体具有以下性质:

A. 对媒体可识别和解释。即各媒体采用各自不同的结构来表达所承载的信息,对各种结构能够理解和正确解释。

B. 不同媒体所表达信息的程序不同,所表达的信息量不同。由于多种媒体在承载信息时,有各自不同的形式特征,因此,它们在表达信息时有程序深浅的不同。一般越接近人的自然表达形式的信息越丰实,越抽象化的信息信息量越少,但也越精确。

C. 媒体之间的相互联系也存在着信息,甚至更多的信息,更重要的信息。

D. 媒体可以进行相互转换。即媒体从一种形式转换为另一种形式。但转换中常伴有信息的损失。同时媒体形式的转换也有一定的限制,有些也是不能直接转换的。如把图像转换成声音即是如此。

正确答案: D. 任何媒体都可以直接进行相互转换。

点评:

本试题要考查的内容是媒体数据的性质。数据是以数字、字符或图像等可读语言或其他记录方法表示的事实、概念或指令,适用于人或自动装置进行通信、解释或处理。数据本身没有意义,通常需要在一定的语义环境中才有意义。本试题中的媒体数据应该是指文字数据、声音数据、图像数据和视像数据,或它们组合而成的多媒体数据。媒体数据的特性通常认为是数据的冗余性,声音、图像和视像数据占据的存储空间大、网上传输时间长,声音和视像数据的传输要求很高的实时性和连续性等。

围绕“媒体数据性质”,反复阅读四个“关于媒体数据性质的描述”和试题分析,其中有些提法或论断对笔者来说很费解,留下许多似是而非的困惑。例如,在“C. 媒体之间的关系也代表着信息”和“C. 媒体之间的相互联系也存在着信息,甚至更多的信息,更重要的信息”的论断中,“媒体之间的相互关系”是指什么关系、这些论断是否成立以及媒体数据是否具有这种性质等都需要笔者进一步学习和请教。

## 4.2.2 多媒体计算技术

### 1. 数据压缩

#### 【例 4-2】哈夫曼编码

假设要设计一个无损压缩的编码器,用于压缩只有 7 种 (C1, C2, C3, C4, C5, C6, C7) 颜色值的彩色图像。为计算各种颜色值可能出现的数目,对 100 幅  $256 \times 256$  的彩色图



像做了统计。统计结果是, 颜色值为 C1 的像点数占整个像点数的  $1/2$ , 颜色值为 C2 的像点数占整个像点数的  $1/8$ , 其余的见表 4-4 “7 色图像霍夫曼编码表”。使用霍夫曼 (Huffman) 算法设计得到“代码分配方案 A”和“代码分配方案 B”, 两种方案列在表中。问正确答案是\_\_\_\_\_。

- A. “代码分配方案 A”对, “代码分配方案 B”错
- B. “代码分配方案 A”错, “代码分配方案 B”对
- C. “代码分配方案 A”错, “代码分配方案 B”错
- D. “代码分配方案 A”对, “代码分配方案 B”对

表 4-4 7 色图像霍夫曼编码表

颜色名称	各种颜色像点 占指总像点的比例	代码分配方案 A	代码分配方案 B
C1	1/2	1	0
C2	1/8	011	100
C3	1/8	010	101
C4	1/16	0011	1100
C5	1/16	0010	1101
C6	1/16	0001	1110
C7	1/16	0000	1111

分析:

霍夫曼码编码步骤如下:

① 初始化, 根据像素概率的大小按由大到小顺序对像素进行排序, 如表 4.4 和图 4-3 所示。

② 把概率最小的两个像素组成一个结点, 如图中的 C7 和 C6 组成结点  $p_1$ 。

③ 重复步骤②, 得到结点  $p_2, p_3, \dots, p_6$ , 形成一棵“树”。其中的  $p_6$  称为根结点。

④ 从根结点  $p_6$  开始到每个像素的“树叶”, 从上到下标上“1”(上枝)或者“0”(下枝)。至于哪个为“1”哪个为“0”则无关紧要, 最后的结果仅仅是分配的代码不同, 而代码的平均长度是相同的。

⑤ 从根结点  $p_6$  开始顺着树枝到每个叶子分别写出每个像素的代码。

正确答案: D. “代码分配方案 A”对, “代码分配方案 B”对。

请注意: 该题的代码分配方案不惟一。

【例 4-3】哈夫曼编码

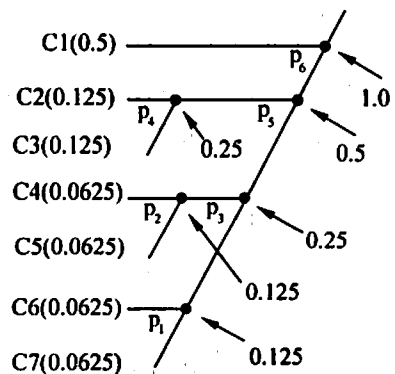


图 4-3 哈夫曼编码树

计算例 4-2 中表示每个像素值 (或者叫做符号) 的平均长度 (指信源的熵)

分析:

(1) 方法 1: 按照信源熵 (Entropy of the Source) 的计算公式计算,

$$\begin{aligned} H(S) &= H(S) = -\sum_{i=1}^N p(a_i) \log_2 [p(a_i)] \\ &= -(1/2 \times \log_2(1/2) + 2 \times 1/8 \times \log_2(1/8) + 4 \times 1/16 \times \log_2(1/16)) \\ &= 1/2 + 2 \times 3/8 + 4 \times 4/16 = 9/4 = 2.25 (\text{位/像素}) \end{aligned}$$

(2) 方法 2: 根据例 4-2 的计算结果, 用分配给每种像素的位数乘其概率之后求和,

$$\begin{aligned} &1 \times 1/2 + 3 \times 1/8 + 3 \times 1/8 + 4 \times 1/16 + 4 \times 1/16 + 4 \times 1/16 + 4 \times 1/16 \\ &= 9/4 = 2.25 (\text{位/像素}) \end{aligned}$$

以上两种计算得到相同的结果。

## 2. 声音技术

### 【例 4-4】声音数字化

从可供选择的答案中, 选择出应该填入下面叙述中\_\_\_内的最确切的解答。

在多媒体音频处理中, 由于人所敏感的声频最高为 (1) 赫兹 (Hz), 因此, 数字声音文件中对音频的采样频率为 (2) 赫兹 (Hz)。对一个双声道的立体声, 保持一秒钟声音, 其波形文件所需的字节数为 (3), 这里假设每个采样点的量化位数为 8 位。

MIDI 文件是最常用的数字音频文件之一, MIDI 是一种 (4), 它是该领域国际上的一个 (5)。

- |               |           |           |          |
|---------------|-----------|-----------|----------|
| (1) A. 50K    | B. 10K    | C. 22K    | D. 44K   |
| (2) A. 44.1K  | B. 20.05K | C. 10K    | D. 88K   |
| (3) A. 22050  | B. 88200  | C. 176400 | D. 44100 |
| (4) A. 语音数字接口 | B. 乐器数字接口 |           |          |
| C. 语音模拟接口     | D. 乐器模拟接口 |           |          |
| (5) A. 控制方式   | B. 管理规范   | C. 通信标准   | D. 输入格式  |

分析:

(1) 人的听觉系统感知的最高频率是 20 000 Hz, 因此似乎正确答案: C. 22K。如果考生的答卷写成 “A~D 都不对”, 判卷的老师也应该算答对。

(2) 声音采样频率是指每秒钟获取声音信号样本的次数, 其大小由奈奎斯特理论 (Nyquist Theory) 和声音信号本身的最高频率决定。奈奎斯特理论指出, 采样频率不应低于声音信号最高频率的两倍, 这样就能把以数字表达的声音还原成原来的声音。采样定律用公式表示为

$$f_s \geq 2f \quad \text{或者} \quad T_s \leq T/2 \quad T_s \leq T/2$$

其中  $f$  为被采样信号的最高频率。

人耳朵 (因人而异) 能听到的声音信号频率范围是 20~20 000 Hz, 为避免高于

20 000 Hz 的高频信号干扰采样, 在进行采样之前, 需要对输入的声音信号进行滤波。考虑到滤波器在 20 000 Hz 的地方大约有 10% 的衰减, 所以可以用 22 000 Hz 的 2 倍频率作为声音信号的采样频率。但是, 为了能够与电视信号同步, PAL 电视的场扫描为 50 Hz, NTSC 电视的场扫描为 60 Hz, 所以取 50 和 60 的公倍数, 因此激光唱盘声音(CD—Audio) 的采样频率选用了 44 100 Hz 作为标准。

根据题意和以上分析, 正确答案: A. 44.1 kHz。

(3) 要回答波形声音文件的大小, 需要知道声音的采样频率(样本/秒)、位数(bits)/样本、声音的通道数和持续时间(秒)。根据题意,

正确答案: B.  $88200=1 \text{ 秒} \times 44\,100 \text{ 样本/秒} \times 8 \text{ 位/样本} \times 2(\text{左右两个通道})/8$

(4) 顾名思义, MIDI (Musical Instrument Digital Interface) 是 B. 乐器数字接口。

(5) MIDI 是在音乐合成器、乐器和计算机之间交换音乐信息的接口标准, 是这些 MIDI 设备使用的一套指令。根据 MIDI 的定义和提供的答案选择, 正确答案: C. 通信标准。

点评:

本题流传很广, 它抓住了要求学生掌握的基础知识, 但题目本身似乎还有可改进的地方, 尤其是下面的两个问题值得讨论。

(1) “人所敏感的声频最高为 (1) 赫兹(Hz)”。人所敏感的最高声音率是 20 000 Hz 还是 22 000 Hz, 这是值得核实的事实。

(2) “MIDI 文件是最常用的数字音频文件之一”。从这个试题的叙述来看, 该试题把 MIDI 文件与数字化的 WAV 声音文件都说成“数字音频文件”, 这种说法欠妥。因为在 MIDI 设备之间的电缆上传送的不是数字声音信号, 而是产生声音或执行某个动作的称为 MIDI 消息的指令, 如播放音符、加大音量和生成音响效果等。MIDI 设备可用来创建、记录和播放音乐的 MIDI 文件, MIDI 文件包含的是乐谱信息, 与播放时间相同的波形文件相比, 其数据量少得多, 因而常用在 Web 网页中作为背景音乐。MIDI 文件的文件扩展名为.mid。

#### 【例 4-5】声音文件格式

PC 机中数字化后的声音有两类表示方式: 一类是波形声音, 一类是合成声音。下列表示方式中, \_\_\_\_ 是一种合成声音文件的后缀。

A. WAV      B. MID      C. RA      D. MP3

分析:

正确回答这个问题的关键是你对这四种声音文件格式含义的了解, 正确答案: B. MID。解释如下:

(1) WAV: 在 Windows 中, 以波形形式存储声音数据的文件格式。以该格式存储的文件的扩展名为.wav。根据不同的采样频率(11 025 Hz, 22 050 Hz 或 44 100 Hz)、是否为立体声以及每个样本的精度(8 位或 16 位), 存储一分钟的声音将占据 630 KB~27 MB

的存储空间。该格式支持各种不同的压缩方法,但最普通的是用 ADPCM 方法,压缩比约为 4:1。

(2) MID: 乐器数字接口 (Musical Instrument Digital Interface, MIDI) 是在音乐合成器、乐器和计算机之间交换音乐信息的接口标准,用于描述这些 MIDI 设备使用的一套指令, MIDI 文件的扩展名是 MID。因为 MIDI 描述的是音乐演奏过程的指令,因此 MIDI 文件比 WAV 文件所需的存储空间小。

(3) MP3: MP3 (MPEG Audio Layer 3) 是使用 MPEG 声音层 3 压缩技术和存储格式的声音文件。在 MPEG-1 Audio 标准中的声音压缩技术分为 1 层, 2 层和 3 层。层 1 典型的压缩比为 1:4, 相应的数据率为 384 Kbps; 层 2 典型的压缩比为 1:6~1:8, 数据率为 256~192Kbps; 层 3 典型的压缩比为 1:10~1:12, 相应的数据率为 128~112 Kbps, 声音质量接近 CD—DA。MP3 文件是目前因特网上最流行的文件, 它的文件扩展名是 .mp3, 可以下载到袖珍 MP3 播放机或计算机上播放。

(4) RA (Real Audio): RealNetworks 公司为因特网和内联网开发的流式声音媒体技术, 包括声音编码和解码技术, 始于 1995 年。使用这种技术开发的软件可把预先录制的或现场的声音实时传送给客户端 (如 Web 浏览器), 而不需要用户事先下载。它采用客户/服务器结构, 服务器 RealServers 将声音数据压缩成 RealAudio 格式的文件, 客户端的 Web 浏览器使用 RealPlayer 插件程序或附加程序, 将从服务器传送过来的数据流解压缩, 用计算机的声音设备输出。RealAudio 文件的扩展名通常为 .ra, .rm 或 .ram。

(5) 部分声音文件扩展名见表 4-5。

表 4-5 部分声音文件扩展名

文件的扩展名	说 明
Au	Sun 和 NeXT 公司的声音文件存储格式 (8 位 $\mu$ 律编码或者 16 位线性编码)
aif(Audio Interchange)	Apple 计算机上的声音文件存储格式
Cmf(Creative Music Format)	声霸 (SB) 卡带的 MIDI 文件存储格式
mct	MIDI 文件存储格式
mff(MIDI Fils Format)	MIDI 文件存储格式
mid(MIDI)	Windows 的 MIDI 文件存储格式
mp2	MPEG Layer I, II
mp3	MPEG Layer III
mod(Module)	MIDI 文件存储格式
Rm(RealMedia)	RealNetworks 公司的流放式声音文件格式
Ra(RealAudio)	RealNetworks 公司的流放式声音文件格式
rol	Adlib 声音卡文件存储格式
snd(sound)	Apple 计算机上的声音文件存储格式



续表

文件的扩展名	说 明
seq	MIDI 文件存储格式
sng	MIDI 文件存储格式
voc(Creative Voice)	声霸卡存储的声音文件存储格式
Wav(Waveform)*	Windows 采用的波形声音文件存储格式
wrk	Cakewalk Pro 软件采用的 MIDI 文件存储格式

#### 【例 4-6】计算机是怎样发声的

实现计算机语音输出有录音重放和 (1) 两种方法。第二种方法是基于 (2) 技术的一种声音产生技术。采用这种方法应预先建立语言参数数据库、发音规则等。

(1) A. 文—语转换      B. 语—文转换      C. 语音编码      D. 语音解码

(2) A. 语音转换      B. 语音合成      C. 语音放大      D. 声音过滤

分析:

在这道试题中,“文—语转换”和“语—文转换”这两个术语对读者可能是比较生疏的。在其他几个术语中,有些是读者比较熟悉的,有些术语虽然不常见,但可以猜出它的基本含义。

“文—语转换 (Text to Speech, TTS)” 是使用计算机把文字转换为声音输出的过程。文语转换的最终目标是要使计算机像人一样输出清晰而又自然的声音,也就是说,根据文章的内容可以不同的情调来朗读任意的文章。TTS 是一个十分复杂的系统,涉及到语言学、语音学、信号处理、人工智能等诸多的学科。尽管现有的 TTS 系统结构各异,转换方法不同,但是基本上可以分成两个相对独立的部分:①文本分析:通过对输入文章进行词法分析和语法分析,甚至进行语义分析,从而抽取音素和韵律等发音信息。②语音合成:使用从文章分析得到的发音信息去控制合成单元的谱特征(音色)和韵律特征(基频、时长和幅度),送入声音合成器(软件或硬件)产生相应的语音输出。

“语—文转换 (Speech to Text)” 实际上是语音识别 (Speech Recognition 或 Voice Recognition), 它是使用计算机把人类声音所携带的信息转换成文字的过程,简称为“语—文转换”。语音识别系统首先把语音数字化,使用各种数学方法抽取声音的特征,然后在编码词典中进行匹配,从而达到识别字、词汇和句子目的。要开发一个用于处理多种不同言语模式和口音的语音识别系统,已经被证实是一项艰巨任务。语音识别从说话人的角度可分为特定人识别和非特定人识别,从口语词汇的组织形式角度可以分为单词识别和连续语音识别。

(1) 在第一个问题中,“语音编码”是声音数据如何表示的问题,而“语音解码”则是如何重构声音数据的问题,它们不是产生声音的方法。根据“文—语转换”和“语—文转换”这两个术语的分析,正确答案: A. 文—语转换。

(2) 顾名思义,语音转换、语音放大和声音过滤都不是声音产生技术。根据对

“文—语转换”的分析。

正确答案: B. 语音合成。

### 3. 图像技术

#### 【例 4-7】颜色表示法

视觉上的颜色可用亮度、色调和饱和度 3 个特征来描述, 饱和度是指颜色的\_\_\_\_\_。

- A. 种数                      B. 纯度                      C. 感觉                      D. 存储器

分析:

饱和度 (saturation) 用来区别颜色明暗程度。当一种颜色渗入其他光的成分愈多时, 颜色愈不饱和。完全饱和的颜色是指没有渗入白光所呈现的颜色, 如仅由单一波长组成的光谱色就是完全饱和的颜色。饱和度在颜色圆上用半径表示, 如图 4-4 (a) 所示, 沿径向方向上的不同颜色具有相同的色调和明度, 但它们的饱和度不同。图 (b) 是在同一半径方向上取出的七种颜色, 它们具有相同的色调和明度, 但具有不同的饱和度, 左边的饱和度最浅, 右边的饱和度最深。

正确答案: B. 纯度。

#### 【例 4-8】RGB 和 CMY 颜色空间

在 RGB 彩色空间中, R (红)、G (绿)、B (蓝) 为三基色, 青色、品红和黄色分别为红、绿、蓝三色的补色。根据相加混色原理, 绿色 + 品红 = \_\_\_\_\_。

- A. 蓝色    B. 黄色    C. 紫色    D. 白色

分析:

该题涉及到 RGB 和 CMY 两个颜色空间和它们之间的相互关系问题。

RGB (Red-Green-Blue, RGB) 颜色空间是使用红、绿、蓝三种基色构成的颜色空间。从原理上说, 任何一种颜色都可用这三种基本颜色光按不同比例混合得到。三种颜色的光越强, 到达我们眼睛的光就越多; 它们的比例不同, 我们看到的颜色也就不同; 没有光到达眼睛, 就是一片漆黑。在这种颜色空间中, 某一种颜色和这三种颜色之间的关系可用下面的式子来描述,

$$\text{颜色} = R(\text{红色的百分比}) + G(\text{绿色的百分比}) + B(\text{蓝色的百分比})$$

如果 R, G, B 都用 1 (100%) 或 0 (0%) 来混合, 就可得到 8 种颜色, 如表 4-6 所示。

CMY (Cyan-Magenta-Yellow) 颜色空间是由青色、品红和黄色这三种减基色组成的颜色空间。从理论上说, 任何一种颜料的颜色都可以由这三种基本颜色的颜料按一定比例混合得到。用这种技术合成颜色的方法又称为相减混色。之所以这样称呼是因为它们混合之后减少了人的眼睛识别颜色所需要的反射光。在相减混色中, 三种颜色的比例不同, 混合后我们看到的颜色也就不同。如果 C, M, Y 分别取 1 (100%) 或 0 (0%) 来

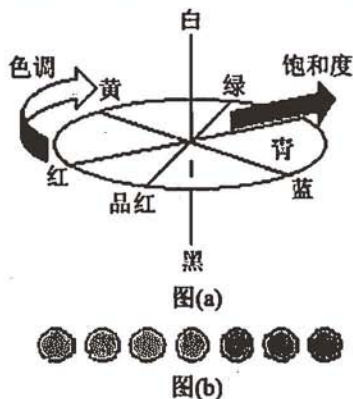


图 4-4 饱和度表示法

混合,就可得到 8 种颜色,如表 4-7 “CMY 相减混色”所示。

通常,我们能够记住的是,在 RGB 颜色空间中等量的 R (红色)、G (绿色) 和 B (蓝色) 相加得到白色,在 CMY 空间中等量的 C (青色)、M (品红色) 和 Y (黄色) 相减得到黑色,而且这两个颜色空间是互补的。对在 RGB 空间中,“绿色+品红”是什么颜色是不容易记住的。

表 4-6 RGB 相加混色

R(红色)	G(绿色)	B(蓝色)	相加色
0	0	0	黑
0	0	1	蓝
0	1	0	绿
0	1	1	青
1	0	0	红
1	0	1	品红
1	1	0	黄
1	1	1	白

表 4-7 CMY 相减混色

C(青色)	M(品红)	Y(黄色)	相减色
0	0	0	白
0	0	1	黄
0	1	0	品红
0	1	1	红
1	0	0	青
1	0	1	绿
1	1	0	蓝
1	1	1	黑

在 CMY 空间中,当 CMY=010 时,生成的颜色是品红色,由此可知,在 RGB 中生成品红色的颜色是 RGB=101。因此,在 RGB 空间中,G (绿色)+M (品红色)就是等量的 R (红色)、G (绿色) 和 B (蓝色) 相加,即 RGB=111,产生的颜色是白色。因此,正确答案: D. 白色。

#### 【例 4-9】图像表示法

在显示存储器中,表示黑白图像的像素最少需 (1) 位。彩色图像可以用 (2) 三基色表示。

- (1) A. 1                      B. 2                      C. 3                      D. 4  
 (2) A. 红黄蓝              B. 红绿蓝              C. 绿黄蓝              D. 红绿黄

分析:

一幅黑白图像、灰度图像或彩色图像都可看成由许多点组成,如图 4-5 所示。图像中的单个点称为像素(pixel),每个像素都有一个值,称为像素值,它表示特定颜色的强度。一个像素值通常用 R, G, B 三个颜色分量表示。

(1) 黑白图像的每个像素值只要用“1”和“0”两个值分别表示“黑”和“白”或相反,因此用需要用二进制的 1 位表示就可以,这种图像也称二值图像。

正确答案: A. 1。

(2) 显示彩色图像用 RGB 颜色空间,因此正确答案: B. 红绿蓝。

#### 【例 4-10】图形和图像的概念



图 4-5 一幅图像由许多像素组成



下列关于在计算机图形图像的描述中,不正确的是\_\_\_\_\_。

- A. 图像都是由一些排成行列的点(像素)组成的,通常称为位图或点阵图
- B. 图像的最大优点是容易进行移动、缩放、旋转和扭曲等变换
- C. 图形是用计算机绘制的画面,也称矢量图
- D. 图形文件中只记录生成图的算法和图上的某些特征点,数据量较小

分析:

本题分析的全文引自“2004 试题分析与解答”,全文如下:

在计算机科学中,图形和图像这两个概念是有区别的:

图像(也称为位图或点阵图, Bit-Map Image)是指由输入设备捕捉的实际场景画面或以数字化形式存储的任意画面。图像都是由一些排成行列的像素组成的,在计算机中的存储格式有 BMP、PCX、TIF、GIF 等,一般数据量都较大。它除了可以表达真实的照片,也可以表现复杂绘画的某些细节,并具有灵活和富于创造力等特点。通常把一幅位图图像考虑为一个矩阵,矩阵中的一个元素(像素)对应图像的一个点,相应的值表示该点的灰度或颜色等级。

图形(也称矢量图形, Vector-Based Image)一般指用计算机绘制的画面,如直线、圆、圆弧、任意曲线和图表等。图形是用一个指令集合来描述的。这些指令用来描述图中线条的形状、位置、颜色等各种属性和参数。

与图像文件不同,在图形文件中只记录生成图的算法和图上的某些特征点。在计算机还原输出时,相邻的特征点之间用特定的很多段小直线连接就形成曲线,若曲线是一条封闭的图形,也可靠着色算法来填充颜色。它的最大优点是容易进行移动、缩放、旋转和扭曲等变换,主要用于表示线框型的图画、工程制图、美术字等。常用的矢量图形文件有 3DS(用于 3D 造型)、DXF(用于 CAD)、WMF(用于桌面出版)等。图形只保存算法和特征点,所以相对于位图的大数据量来说,它占用的存储空间也较小。但由于每次屏幕显示时都需重新计算,故显示速度没有图像快。另外在打印输出和放大时,图形的质量较高而点阵图常会发生失真。

正确答案: B. 图像的最大优点是容易进行移动、缩放、旋转和扭曲等变换。

点评:

本试题涉及三个常见术语,根据试题分析将它们的定义归纳如下供讨论。

图形图像(Graphical Image)是表示图形的位图,即用矢量图表示的图形转换成了用像素表示的图像。

图像(Image)是表示二维场景的数据。数字图像由具有一定高度和宽度的像素组成,每个像素由一位或多位数据组成。这些数据表示该像素的亮度以及可能的颜色值,用于在显示设备上再现场景。图像通常是指通过数字摄像机、图像捕获卡或扫描仪得到的真实世界的场景。

图形(Graphics)是按照数学规则用计算机绘制的矢量图或者用其他设备创建的图



形, 如工程图和结构图。图形有如图 4-6 所示的两种类型, 用图形输入设备或绘图软件 (如 Adobe Illustrator) 创建的图形是矢量图, 用扫描仪创建的图形是位图形, 也称光栅图形。

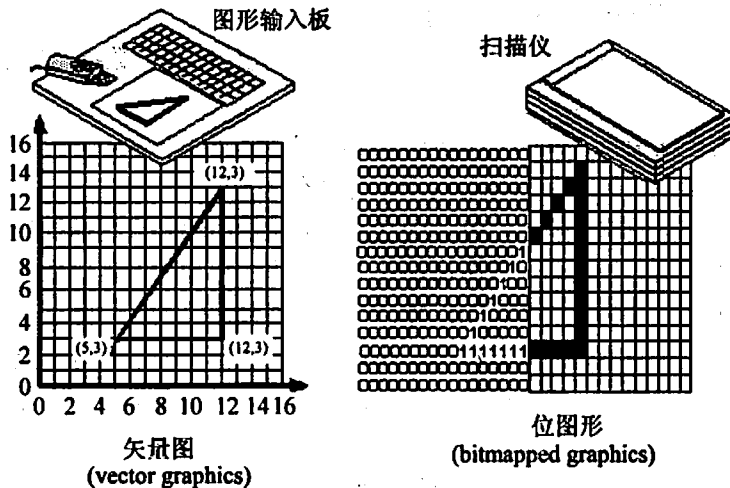


图 4-6 矢量图和位图形

#### 【例 4-11】图像表示法

以像素点阵描述的图像称为\_\_\_\_\_。

- A. 位图      B. 投影图      C. 矢量图      D. 几何图

分析:

位图 (bitmap) 应该用像素值阵列表示的图形或图像。如图 4-7 所示, 图 4-7 (a) 是用像素值表示的图像 (边缘较光滑), 图 4-7 (b) 是用矢量图生成的图形转换成用像素表示的图形图像 (边缘有锯齿)。对位图进行操作时, 只能对图中的像素进行操作, 而不能把位图中的物体可作为独立实体进行操作。

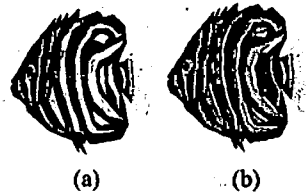


图 4-7 位图

矢量图 (Vector Graphics) 是根据数学规则描述用生成的图。一幅图用数学描述的点、线、弧、曲线、多边形和其他几何实体等对象和几何位置来表示, 创建的图是对象的集合而不是点或像素模式的图。

正确答案: A. 位图。

#### 【例 4-12】文件格式

计算机中声音、图形图像信息都是以文件的形式存储的, 它们的文件格式有许多种, 可以用扩展名来识别, 常见的文件格式扩展名有:

- ① BMP    ② AIF    ③ JPG    ④ WAV    ⑤ GIF    ⑥ VOC

其中表示声音文件的 (1), 表示图形图像文件的有 (2)。

- (1) A. ① ② ③    B. ① ③ ⑤    C. ④ ⑤ ⑥    D. ② ④ ⑥  
 (2) A. ① ② ③    B. ① ③ ⑤    C. ④ ⑤ ⑥    D. ② ④ ⑥

分析:

与存储文字文件一样, 存储图像数据也需要有存储格式。文件格式可大致分成三类: 文本文件格式、声音文件格式和图像文件格式。至今为止, 有文献记载的就不少于 100 多种, 要记住它们几乎是不可能的。但对常见的文件格式要有意识地记一下, 以便于我们的学习和工作。常见的图像文件格式包括 BMP, GIF, JPG, PNG 等, 常见的声音文件格式包括 WAV, MID, MP3, RA, VOC 等, 数据压缩文件包括 ZIP, RAR, TXT, DOC 等。

试题中的 AIF/ aif (Audio Interchange) 格式是 Apple 计算机上的声音文件存储格式, VOC/ voc (Creative Voice) 是声霸卡存储的声音文件存储格式。

(1) 正确答案: D. ② ④ ⑥, 即 AIF, WAV, VOC

(2) 正确答案: B. ① ③ ⑤, 即 BMP, JPG, GIF

下面就试题中的几种图像文件存储格式做一个比较详细的介绍。

① **BMP**: 位图文件 (Bitmap-File, BMP) 格式是 Windows 采用的图像文件存储格式, 在 Windows 环境下运行的所有图像处理软件都支持这种格式。Windows 3.0 以前的 BMP 位图文件格式与显示设备有关, 因此把它称为设备相关位图 (Device-Dependent Bitmap, DDB) 文件格式。Windows 3.0 以后的 BMP 位图文件格式与显示设备无关, 因此把这种 BMP 位图文件格式称为设备无关位图 (Device-Independent Bitmap, DIB) 格式, 目的是为了让 Windows 能够在任何类型的显示设备上显示 BMP 位图文件。BMP 位图文件默认的文件扩展名是 BMP 或者 bmp。

② **JPG**: JPG 是使用 JPEG Joint Photographic Experts Group 文件交换格式存储的编码图像文件的扩展名。JPEG 委员会在制定 JPEG 标准时, 定义了许多标记 (marker) 用来区分和识别图像数据及其相关信息, 但笔者没有找到 JPEG 委员会对 JPEG 文件交换格式的明确定义。直到 1998 年 12 月从分析网上具体的 JPG 图像来看, 使用比较广泛的还是 JPEG 文件交换格式 (JPEG File Interchange Format, JFIF), 版本号为 1.02。这是 1992 年 9 月由在 C-Cube Microsystems 公司工作的 Eric Hamilton 提出的。JPEG 文件使用的颜色空间是 1982 年推荐的电视图像信号数字化标准 CCIR 601, 现改为 ITU-R BT.601。在这个彩色空间中, 每个分量、每个像素的电平规定为 255 级, 用 8 位代码表示。RGB 和 YCbCr 空间使用如下的转换关系:

从 RGB 转换成 YcbCr: YCbCr 分量, [0, 255], 可直接从用 8 位表示的 RGB 分量计算得到:

$$\begin{cases} Y = 0.299 R + 0.587 G + 0.114 B \\ Cb = -0.1687 R - 0.3313 G + 0.5 B + 128 \\ Cr = 0.5 R - 0.4187 G - 0.0813 B + 128 \end{cases}$$

需要注意的是不是所有图像文件格式都按照  $R_0 G_0 B_0, \dots, R_n G_n B_n$  的次序存储样本数据, 因此在 RGB 文件转换成 JFIF 文件时需要首先验证 R, G, B 的次序。

从 YCbCr 转换成 RGB: RGB 分量可直接从 YCbCr 分量, [0, 255], 计算得到:

$$\begin{cases} R = Y + 0 + 1.40200 (Cr - 128) \\ G = Y - 0.34414 (Cb - 128) - 0.71414 (Cr - 128) \\ B = Y + 1.77200 (Cb - 128) + 0 \end{cases}$$

在 JFIF 文件格式中, 图像样本的存放顺序是从左到右和从上到下。这就是说 JFIF 文件中的第一个图像样本是图像左上角的样本。

此外, 微处理机中的数据存放顺序有正序 (Big Endian) 和逆序 (Little Endian) 之分。正序存放就是高字节存放在前低字节在后, 而逆序存放就是低字节在前高字节在后。例如, 十六进制数为 A02B, 正序存放就是 A02B, 逆序存放就是 2BA0。摩托罗拉 (Motorola) 公司的微处理器使用正序存放, 而英特尔 (Intel) 公司的微处理器使用逆序。JPEG 文件中的字节是按照正序排列的。

③ GIF: 图形文件交换格式 (Graphics Interchange Format, GIF) 是 20 世纪 80 年代 CompuServe 公司开发的图像文件存储格式, 支持的图像颜色可达 256 种。1987 年开发的 GIF 文件格式版本号是 GIF87a, 1989 年进行了扩充, 扩充后的版本号定义为 GIF89a。GIF 图像文件以数据块为单位来存储图像的相关信息。一个 GIF 文件由表示图像的数据块、数据子块以及显示图像的控制信息块组成, 称为 GIF 数据流。数据流中的所有控制信息块和数据块都必须在文件头和文件结束块之间。GIF 文件格式采用了 LZW (Lempel—Ziv Walch) 压缩算法来压缩图像数据, 定义了允许用户为图像设置背景的透明属性。用 GIF 文件格式存储的文件大小与实际使用的颜色数有关。此外, GIF 文件格式可在一个文件中存放多幅彩色图像。如果在 GIF 文件中存放有多幅图像, 显示的效果就是动画。

表 4-8 和表 4-9 列出了部分位映像图和矢量图文件格式的后缀供参考。

表 4-8 位映像图格式/光栅图光栅 (Bitmapped Formats / Raster Graphics)

后缀	文件名称	后缀	文件名称
AG4	Access G4 document imaging	JFF	JPEG (JFIF)
ATT	AT&T Group IV	JPG	JPEG
BMP	Windows & OS/2	KFX	Kofax Group IV
CAL	CALS Group IV	MAC	MacPaint
CIT	Intergraph scanned images	MIL	Same as GP4 extension
CLP	Windows Clipboard	MSP	Microsoft Paint
CMP	Photomatrix G3/G4 scanner format	NIF	Navy Image File

后缀	文件名称	后缀	文件名称
CMP	LEAD Technologies	PBM	Portable bitmap
CPR	Knowledge Access	PCD	PhotoCD
CT	Scitex Continuous Tone	PCX	PC Paintbrush
CUT	Dr. Halo	PLX	Inset Systems (HiJaak)
DBX	DATABEAM	PNG	Portable Network Graphics
DX	Autotrol document imaging	PSD	Photoshop native format
ED6	EDMICS (U.S. DOD)	RAS	Sun
EPS	Encapsulated PostScript	RGB	SGI
FAX	Fax	RIA	Alpharel Group IV document imaging
FMV	FrameMaker	RLC	Image Systems
GED	Arts & Letters	RLE	Various RLE-compressed formats
GDF	IBM GDDM format	RNL	GTX Runlength
GIF	CompuServe	SBP	IBM StoryBoard
GP4	CALS Group IV — ITU Group IV	SGI	Silicon Graphics RGB
GX1	Show Partner	SUN	Sun
GX2	Show Partner	TGA	Targa
ICA	IBM IOCA (see MO:DCA)	TIF	TIFF
ICO	Windows icon	WPG	WordPerfect image
IFF	Amiga ILBM	XBM	X Window bitmap
IGF	Inset Systems (HiJaak)	XPM	X Window pixelmap
IMG	GEM Paint	XWD	X Window dump

表 4-9 矢量图格式 (Vector Graphics Formats)

后缀	文件名称	后缀	文件名称
3DS	3D Studio	GEM	GEM proprietary
906	Calcomp plotter	G4	GTX RasterCAD - scanned images into vectors for AutoCAD
AI	Adobe Illustrator	IGF	Inset Systems (HiJaak)
CAL	CALS subset of CGM	IGS	IGES
CDR	CorelDRAW	MCS	MathCAD
CGM	Computer Graphics Metafile	MET	OS/2 metafile
CH3	Harvard Graphics chart	MRK	Informative Graphics markup file
CLP	Windows clipboard	P10	Tektronix plotter (PLOT10)
CMX	Corel Metafile Exchange	PCL	HP LaserJet
DG	Autotrol	PCT	Macintosh PICT drawings
DGN	Intergraph drawing format	PDW	HiJaak
DRW	Micrografx Designer 2.x, 3.x	PGL	HP plotter



续表

后缀	文件名称	后缀	文件名称
DS4	Micrografx Designer 4.x	PIC	Variety of picture formats
DSF	Micrografx Designer 5.x	PIX	Inset Systems (HiJaak)
DXF	AutoCAD	PLT	HPGL Plot File (HPGL2 has raster format)
DWG	AutoCAD	PS	PostScript Level 2
EMF	Enhanced metafile	RLC	Image Systems "CAD Overlay ESP" vector files overlaid onto raster images
EPS	Encapsulated PostScript	SSK	SmartSketch
ESI	Esri plot file (GIS mapping)	WMF	Windows Metafile
FMV	FrameMaker	WPG	WordPerfect graphics
GCA	IBM GOCA	WRL	VRML

#### 4. 数字电视

##### 【例 4-13】彩色电视制式

20 世纪 70 年代出现的 LD (Laser—Disc) 模拟视盘系统使用恒定线速度 (Constant Linear Velocity, CLV) 或者恒定角速度 (Constant Angular Velocity, CAV) 来驱动盘片, 盘的直径有 12 英寸和 8 英寸两种, 使用两个记录面。现有一张播放两个小时电影的 LD 盘, 每一面存有 108000 帧的图像。推断这张盘上的电视制式是\_\_\_\_\_。

A. PAL            B. NTSC            C. SECAM            D. 电影

分析:

根据试题给的条件, 要正确回答这个问题首先要搞清楚这些电视制式和电影的帧速率, 然后与计算得到的帧速率进行比较就可判断属于哪种制式。这片光盘盘上电视的帧速率为

$$108000 \text{ 帧/面} \times 2 \text{ 面} / (2 \times 60 \times 60) \text{ 秒} = 30 \text{ 帧/秒}$$

PAL 制的帧速率为 25 帧每秒, NTSC 制的帧速率为 30 帧每秒, SECAM 制的帧速率为 25 帧每秒, 电影的帧速率为 24 帧每秒, 因此正确答案: B. NTSC。

为加深对这道试题的理解, 现将试题中涉及的电视制式简介如下:

(1) PAL 制: PAL (Phase-Alternative Line) 制彩色电视制式称为逐行倒相彩色电视制式。这是中国大陆和除法国和俄罗斯外的大多数西欧国家采用的彩色电视广播标准, 它是 1967 年开发的电视制式。除高清晰度电视 (HDTV) 之外, 它是世界上现行三种彩色电视制式 (NTSC, PAL 和 SECAM) 之一。PAL 电视制式的主要特性如下: 图像的宽高比为 4:3, 625 条扫描线, 隔行扫描, 25 帧图像每秒, 4 MHz 的视像带宽, 使用 YUV 颜色模型, 色度信号用正交幅度调制 (QAM), 声音用调频制 (FM), 总的电视通道带宽为 8 MHz。

(2) NTSC 制: 美国国家电视系统委员会 (National Television Systems Committee, NTSC) 在 20 世纪 50 年代初期制定的彩色电视广播标准。除高清晰度电视 (HDTV) 之

外,它是世界上现行三种彩色电视制式(NTSC、PAL和SECAM)之一。该标准被称为正交平衡调幅制,是一个兼容黑白电视信号的彩色电视信号编码系统,使得彩色广播信号可以被黑白电视接收设备接收。NTSC电视的主要特性如下:图像的宽高比为4:3,525条扫描线,隔行扫描,30帧图像每秒,4 MHz的视像带宽,使用YIQ信号,色度信号用正交幅度调制(QAM),声音用调频制(FM),总的电视通道带宽为6 MHz。美国、加拿大等大部分西半球国家以及日本、韩国、菲律宾和我国台湾采用这种制式。

(3) SECAM制:SECAM是法文Sequential Couleur Avec Memoire的缩写。SECAM制是法国开发的彩色电视广播标准,称为顺序传送彩色与存储制式。这种制式与PAL制具有相同的扫描线数(625线每帧)、帧速率(25帧每秒,50场每秒)和图像格式(4:3),它们之间的主要差别是SECAM中的色度信号是频率调制(FM),而PAL和NTSC则采用正交幅度调制(QAM),而且它的两个色差信号:红色差( $R'-Y'$ )和蓝色差( $B'-Y'$ )信号是按行的顺序传输的。SECAM电视信号带宽位6 MHz,总带宽为8 MHz。法国、俄罗斯、东欧和中东等约有65个地区和国家使用这种制式。

**【例4-14】视像的特性**

若电视图像序列中两帧相邻图像之间存在着极大的相关性,则这种相关性称为\_\_\_\_\_冗余。

A. 空间 B. 时间 C. 视觉 D. 信息熵

分析:

正确答案: B. 时间。

本题考查的知识是视像数据的特性。一个事实是图像数据中有许多重复或相关的数据,用数学方法来表示这些重复数据就可以减少数据量。在空间方向上,一幅图像中的像素值之间存在着相关性,称为“空间冗余”,可用变换编码、预测编码等方法来减小它们之间的相关性;在时间方向上,相邻帧图像数据之间存在相关性,称为“时间冗余”,可用帧间预测、移动补偿等方法来减小它们之间的相关性。另一个事实是人的眼睛对图像细节和颜色的辨认有一个极限,这个特性称为“视觉冗余”,如果把超过极限部分的信息去掉,这也就达到压缩数据的目的。

这两个基本事实都可用来压缩视像数据。利用前一个事实的压缩技术叫做无损数据压缩技术,利用后一个事实的压缩技术叫做有损数据压缩技术。实际的图像和视像数据压缩是综合使用各种有损和无损压缩技术来实现的。

“信息熵冗余”是什么?按照“2004 试题分析与解答”,“信息熵冗余”的定义全文如下:

信息熵是指一组数据所携带的信息量。它一般定义为:

$$H = - \sum_{i=0}^{N-1} P_i \log_2 p_i$$

其中N为数据类数或码元个数, $p_i$ 为码元 $Y_i$ 发生的概率。由定义,为使单数据量d

接近于或等于  $H$ ，应设：

$$d = \sum_{i=0}^{N-1} P_i b(y_i)$$

其中， $b(Y_i)$  是分配给码元  $Y_i$  的比特数，理论上应取  $b(Y_i) = -\log_2 p_i$ ，实际上在应用中很难估计出  $\{p_0, p_1, \dots, p_{n-1}\}$ 。因此一般取

$$b(y_0) = b(y_1) = \dots = b(y_{N-1})$$

这样所得的  $d$  必然大于  $H$ ，由此带来的冗余称为信息熵冗余或编码冗余。

根据上面的引文，本题“信息熵冗余”的概念就是指信源熵（Source Entropy）的概念。下面对几个术语加以补充说明。

(1) 信息量 (Information Content)：具有确定概率事件的信息的定量度量，在数学上定义为

$$I(x) = \log_2 [1/p(x)] = -\log_2 p(x)$$

其中， $p(x)$  是事件  $x$  出现的概率。例如，假说  $X = \{a, b, c\}$  是由 3 个事件构成的集合， $p(a) = 0.5$ ， $p(b) = 0.25$ ， $p(c) = 0.25$  分别是事件  $a, b$  和  $c$  出现的概率，这些事件的信息量分别为

$$I(a) = \log_2 [1/(0.50)] \text{ Sh} = 1 \text{ Sh};$$

$$I(b) = \log_2 [1/(0.25)] \text{ Sh} = 2 \text{ Sh};$$

$$I(c) = \log_2 [1/(0.25)] \text{ Sh} = 2 \text{ Sh}.$$

对一个等概率事件的集合，每个事件的信息量等于该集合的决策量。

(2) 熵 (Entropy)：在有限的互斥和联合穷举事件的集合中，事件的信息量的平均值。某个事件  $i$  的信息量为  $I(x_i) = -\log_2 p(x_i)$ ，其中  $p(x_i)$  为事件  $i$  出现的概率， $0 < p(x_i) \leq 1$ ，表示某一事件出现的消息越多，事件发生的可能性就越小，数学上就是概率越小。熵用数学方法表示为

$$H(X) = \sum_{i=1}^n p(x_i) I(x_i) = -\sum_{i=1}^n p(x_i) \log_2 p(x_i)$$

其中， $X = \{x_1, \dots, x_n\}$  是事件  $x_i (i = 1, 2, \dots, n)$  的集合，并满足

$$\sum_{i=1}^n p(x_i) = 1$$

用在数据压缩中，熵是消息中用符号表示的平均信息量，表示非冗余的、不可压缩的信息量，或者说数据可压缩到什么程度。例如，假说  $X = \{a, b, c\}$  是由 3 个符号构成的集合，符号  $a, b$  和  $c$  出现的概率分别为  $p(a) = 0.5$ ， $p(b) = 0.25$ ， $p(c) = 0.25$ ，它们的熵  $I(x_i) = -p(x) \log_2 p(x_i)$  分别等于 0.5，0.5，0.5，表示对符号进行编码时所需要的最少位 (bit) 数，这个集合的熵为

$$H(X) = p(a) I(a) + p(b) I(b) + p(c) I(c) = 1.5 \text{ Sh}$$

(3) 香农 (Shannon, Sh)：用以 2 为底的对数表示的信息度量单位。例如，8 位字符

集的决策量等于  $3(\log_2 8 = 3)$  香农。香农通常被称为位 (bit)。

(4) 决策量 (Decision Content): 在有限数目的互斥事件集合中, 事件数的对数值, 在数学上表示为

$$H_0 = \log n$$

其中,  $n$  是事件数。对数的底数决定决策量的单位, 通常使用的单位如下:

- ① Sh (Shannon): 用于以 2 为底的对数
- ② Nat (Natural Unit): 用于以  $e$  为底的对数
- ③ Hart (Hartley): 用于以 10 为底的对数

它们之间的转换关系如下:

$$\begin{aligned} 1 \text{ Sh} &= 0.693 \text{ Nat} = 0.301 \text{ Hart} \\ 1 \text{ Nat} &= 1.433 \text{ Sh} = 0.434 \text{ Hart} \\ 1 \text{ Hart} &= 3.322 \text{ Sh} = 2.303 \text{ Nat} \end{aligned}$$

决策量与事件出现的概率无关。例如, 假说  $\{a, b, c\}$  是 3 个事件组成的一个集合, 它的决策量为:

$$\begin{aligned} H_0 &= (\log_2 3) \text{ Sh} = 1.580 \text{ Sh} \\ &= (\log_e 3) \text{ Nat} = 1.098 \text{ Nat} \\ &= (\log_{10} 3) \text{ Hart} = 0.477 \text{ Hart} \end{aligned}$$

#### 【例 4-15】视像数字化

在 YUV 彩色空间中对 YUV 分量进行数字化, 对应的数字化位数通常采用 Y:U:V = \_\_\_\_。

- A. 8:4:2    B. 8:4:4    C. 8:8:4    D. 4:8:8

分析:

本试题分析的全文引自“2004 试题分析与解答”, 全文如下:

【现代彩色电视系统中, 通常采用三管彩色摄像机或彩色 CCD 摄像机, 把摄得的彩色图像信号经分色棱镜分成 R0、G0、B0 三个分量的信号; 分别经放大和校正得到三基色, 再经过矩阵变换电路得到亮度信号 Y、色差信号 R-Y 和 B-Y, 最后发送端将 Y、R-Y 和 B-Y 三个信号进行编码, 用同一信道发送出去, 这就是常用的 YUV 彩色空间。

在多媒体计算机中采用了 YUV 彩色空间, 数字化后通常为 Y:U:V = 8:4:4 或者是 Y:U:V = 8:2:2。】

点评:

选择答案采用这种数值表示采样格式即使 google 都没有找到, 原试题选择答案可改写成如下数值似乎比较好,

- A. 4:2:1    B. 4:2:2    C. 4:4:2    D. 2:4:4

对照表 4-10, 不难看出正确答案: B. 4:2:2。



表 4-10 ITU-R BT.601 彩色电视数数字化参数摘要

采样格式	信号形式	采样频率	样本数/扫描行		数字信号取值
			NTSC	PAL	范围(A/D)
	Y	13.5	858(720)	864(720)	220级(16~235)
4:2:2	Cr	6.75	429(360)	432(360)	225级(16~240)
	Cb	6.75	429(360)	432(360)	(128 ± 112)
	Y	13.5	858(720)	864(720)	220级(16~235)
4:4:4	Cr	13.5	858(720)	864(720)	225级(16~240)
	Cb	13.5	858(720)	864(720)	(128 ± 112)

由于视像数字化是非常基础的问题,从试题本身和对此试题的分析来看,似乎有必要对此做进一步的解释。

在大多数情况下,数字电视系统都希望用彩色分量来表示图像数据,如用 YCbCr, YUV, YIQ 或 RGB 彩色分量,因此,电视图像数字化常用“分量数字化(Component Digitization)”这个术语。电视图像数字化常用的方法有两种:(1)先从复合彩色电视图像中分离出彩色分量,然后数字化。(2)首先用一个高速 A/D 转换器对彩色全电视信号进行数字化,然后在数字域中进行分离,以获得所希望的 YCbCr, YUV, YIQ 或 RGB 分量数据。

试验表明,使用下面介绍的子采样格式,人的视觉系统对采样前后显示的图像质量没有感到有明显差别。目前使用的子采样格式有如下几种(见图 4-8):

(1) 4:4:4 这种采样格式不是子采样格式,它是指在每条扫描线上每 4 个连续的采样点取 4 个亮度 Y 样本、4 个红色差 Cr 样本和 4 个蓝色差 Cb 样本,这就相当于每个像素用 3 个样本表示。

(2) 4:2:2 这种子采样格式是指在每条扫描线上每 4 个连续的采样点取 4 个亮度 Y 样本、2 个红色差 Cr 样本和 2 个蓝色差 Cb 样本,平均每个像素用 2 个样本表示。

(3) 4:1:1 这种子采样格式是指在每条扫描线上每 4 个连续的采样点取 4 个亮度 Y 样本、1 个红色差 Cr 样本和 1 个蓝色差 Cb 样本,平均每个像素用 1.5 个样本表示。

(4) 4:2:0 这种子采样格式是指在水平和垂直方向上每 2 个连续的采样点上取 2 个亮度 Y 样本、1 个红色差 Cr 样本和 1 个蓝色差 Cb 样本,平均每个像素用 1.5 个样本表示。

#### 【例 4-16】视像

MPEG-1 编码器输出影视数据的数据率大约为(1)。PAL 制下其图像亮度信号的分辨率为(2),帧速为(3)。

- (1) A. 128 Kb/s      B. 320Kb/s      C. 1.5 Mb/s      D. 15 Mb/s  
 (2) A. 352×288      B. 576×352      C. 720×576      D. 1024×720  
 (3) A. 16 帧/秒      B. 25 帧/秒      C. 30 帧/秒      D. 50 帧/秒

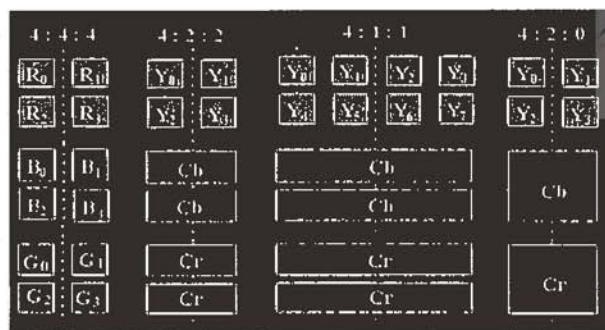


图 4-8 彩色图像 4 种子采样格式的图解说明

分析:

(1) 制定 MPEG—1 标准时, CD—ROM 存储视像和声音时的总数据传输率只有 1.401600 Mb/s, 而当时的网络速度也不高, 因此正确答案: C. 1.5 Mb/s

(2) 该题要回答的是“PAL 制下其图像亮度信号的分辨率”, 联系到题目的前半部分, 或者把题目改为“输入到 MPEG-1 编码器的 PAL 图像亮度信号的分辨率”, 毫无疑问可判断正确答案: A.  $352 \times 288$ 。

(3) 正确答案: B. 25 帧/秒。

试题的详细分析请见 MPEG—1, MPEG—2, MPEG—4 和 MPEG—7 的介绍。

点评:

在闭卷考试前提下, 要正确回答这个问题, 可能只能靠平时阅读文件时有意识地记忆一些重要特性和参数。为帮助同学系统地了解 MPEG 专家组开发的标准, 现将几个标准的要点归纳如下。

① MPEG—1: 1992 年 MPEG 专家组发布的第一个数字电视编码标准, 包括图像数据和声音数据的编码。编码图像对象是数字化后的标准图像交换格式或称为源输入格式的电视, 即 NTSC 制为  $352 \text{ 像素} \times 240 \text{ 行}$  每帧  $\times 30 \text{ 帧每秒}$ , PAL 制为  $352 \text{ 像素} \times 288 \text{ 行}$  每帧  $\times 25 \text{ 帧每秒}$ , 压缩的输出速率定义在 1.5 Mbps 以下, 声音接近 CD—DA 的质量。这个标准主要是针对当时具有这种数据传输率的 CD—ROM 和网络而开发的, 用于在 CD—ROM 上存储数字影视和在网络上传输数字影视。MPEG—1 标准号为 ISO/IEC 11172。

② MPEG—2: MPEG 专家组发布的第二个数字电视编码标准, 用于包括 DVD 和 HDTV 电视在内的高质量图像和声音编码。从 1990 年开始研究的 MPEG—2 定义了大范围的数据压缩率、多声道的声音、宽范围的帧尺寸以及对顺序扫描和隔行扫描的支持等特性。初期的编码对象主要是按照 BT.601 标准数字化后的电视图像, 即  $704 \times 576 \text{ 像素}$  25 帧每秒的 PAL 制电视和  $704 \times 480 \text{ 像素}$  30 帧每秒的 NTSC 制电视, 后来扩展到了 HDTV 电视。MPEG—2 和 MPEG—1 的基本编码算法相同, 如它们都定义了帧图

像 (I), 预测图像 (P) 和双向预测图像 (B) 的编码方法。为适应宽范围的应用, MPEG—2 引入了电视等级和配置类型的概念, 对每种配置类型定义一套算法, 而对每个等级指定一套参数范围, 如图像大小、扫描方式、帧速率和位速率等。MPEG—2 标准号为 ISO/IEC 13818。

③ MPEG—4: MPEG 专家组从 1993 年启动开发的多媒体通信标准, 为可视电话和多媒体应用提供低数据率 (低于 1.5 Mb/s) 的高质量视听数据编码方法和交互播放工具, 并要在异构网络环境下能够实现频繁交互的通信。MPEG—4 为此引入了视听对象编码的概念, 扩充了编码的数据类型, 即由自然数据对象扩展到计算机生成的合成数据对象, 采用了合成对象与自然对象混合编码算法。在实现交互功能和重用对象中引入了组合、合成和编排等重要概念。MPEG—4 标准号为 ISO/IEC 14496。

④ MPEG—7: MPEG 专家组从 1996 年启动开发的多媒体内容描述接口标准, 其目标是描述多媒体信息及它们之间的关系, 以便于信息的检索。这些多媒体信息包括用文字、图像、图形、三维模型、声音和视像等传播媒体表示的信息, 以及它们在多媒体演示中的组合关系。在某些情况下, 还可包括表示面部特性和个人特性的信息。与其他 MPEG 标准一样, MPEG—7 是为满足特定需求而制定的视听信息标准, 它建立在其他标准之上, 如 MPEG—4 中使用的形状描述符、MPEG—1 和 MPEG—2 中使用的移动矢量等。MPEG—7 的应用领域包括数字图书馆 (如图像目录、音乐词典等)、多媒体目录服务 (如黄页)、广播媒体选择 (如无线电频道, TV 频道等)、多媒体编辑 (如个人电子新闻服务、多媒体创作等)。潜在的应用领域包括教育、娱乐、新闻、旅游、医疗、购物等。

### 4.2.3 多媒体存储技术

在过去的程序员和设计者的试题中, 虽然多媒体存储技术方面的试题不多, 但似乎这方面试题有增加的趋势。

#### 【例 4-17】光盘的光道

CD 光盘上记录信息的轨迹叫光道, 信息存储在\_\_\_\_光道上。

- A. 一条圆形
- B. 多条同心环形
- C. 一条渐开的螺旋形
- D. 多条螺旋形

分析:

CD 盘光道的结构与磁盘磁道的结构不同, 它的光道不是同心环光道, 而是螺旋型光道。CD 盘转动的角速度在光盘的内外区是不同的, 而它的线速度是恒定的, 就是光盘的光学读出头相对于盘片运动的线速度是恒定的, 通常用 CLV (Constant Linear Velocity) 表示。由于采用了恒定线速度, 所以内外光道的记录密度 (比特数/每英寸)

可以做到一样,这样盘片就得到充分利用,可以达到它应有的数据存储容量,但随机存储特性变得较差,控制也比较复杂。

CD 光盘上只有一条物理光道,其长度大约为 5 公里。一条物理光道可分成多条逻辑光道。

正确答案: C. 一条渐开的螺旋形。

#### 【例 4-18】光盘容量

DVD-ROM 光盘最多可存储 17 GB 的信息,比 CD-ROM 光盘的 650 MB 大了许多。DVD-ROM 光盘是通过\_\_\_\_来提高存储容量的。

- A. 减小读取激光波长,减小光学物镜数值孔径
- B. 减小读取激光波长,增大光学物镜数值孔径
- C. 增大读取激光波长,减小光学物镜数值孔径
- D. 增大读取激光波长,增大光学物镜数值孔径

分析:

常规的 CD 播放机和 CD-ROM 驱动器采用波长为 780 nm 的不可见红外光来读出盘上的信息。为了把光道距离和信息记录凹凸坑的长度和宽度做得更小,DVD 刻录机和播放机就需要采用波长更短的激光源,这是因为光学读出头的分辨率和激光波长成正比。近年来有许多有关蓝色激光器的报道,但是要把它用到 DVD 上还有漫长的路要走。更现实的技术是使用波长为 635/650 nm 的激光源来代替在 CD 驱动器中使用的 780 nm 红外光激光源。

常规的 CD 播放机和 CD-ROM 驱动器的光学读出头的数值孔径(Numerical Aperture, NA)为 0.45。为了提高盘片反射光的能力,也就是提高光学读出头的分辨率,在 DVD 驱动器中把 NA 由现在的 0.45 加大到 0.6,这样可以产生直径比较小的聚焦激光束。使用短波长的激光源和数值孔径比较大的光学元件之后,最小凹凸的长度可以从 0.83  $\mu\text{m}$  减小到 0.4  $\mu\text{m}$ ,而光道间距从 1.6  $\mu\text{m}$  减小到 0.74  $\mu\text{m}$ ,与 CD-ROM 相比,总的容量可以提高到 4.486 倍。

正确答案: B. 减小读取激光波长,增大光学物镜数值孔径。

此外,用于提高存储容量的其他技术见表 4-11。

表 4-11 HDCD 技术摘要

光盘参数	DVD	CD	容量增益
盘片直径	120 mm	120 mm	
盘片厚度	0.6 mm /面	1.2 mm /面	
减小激光波长	635/650 nm	780 nm	
加大 N.A.(数值孔径)	0.6	0.45	4.486 = (1.6*0.83)/ (0.74*0.40)
减小光道间距	0.74 $\mu\text{m}$	1.6 $\mu\text{m}$	



续表

光盘参数	DVD	CD	容量增益
减小最小凹凸坑长度	0.4 $\mu\text{m}$	0.83 $\mu\text{m}$	
减小纠错码的长度	RSPC	CIRC	
修改信号调制方式	8-16	8-14 加 3	$1.0625 = 17/16$
加大盘片表面的利用率	86.6 $\text{cm}^2$	86 $\text{cm}^2$	$1.019 = 86.6/86$
减小每个扇区字节数	2048/2060 字节/扇区	2048/2352 字节/扇区	$1.142 = 2352/2060$

#### 4.2.4 多媒体网络应用

##### 【例 4-19】 ADSL

ADSL 对应的中文术语是 (1)，它的两种 Internet 接入方式是 (2) 接入。

- (1) A. 分析数字系统层                      B. 非对称数字线  
C. 非对称数字用户线                      D. 异步数字系统层
- (2) A. 固定接入和虚拟拨号                B. 专线接入和 VLAN  
C. 固定接入和 VLAN                      D. 专线接入和虚拟拨号

分析:

本试题分析的全文引自“2004 试题分析与解答”。

(1)【ADSL 的全称为 Asymmetric Digital Subscriber Line, 翻译成中文为非对称数字用户线】。

正确答案: C. 非对称数字用户线。

(2)【它的接入类型有两种: 专线入网方式: 用户拥有固定的静态 IP 地址, 24 小时在线。② 虚拟拨号入网方式: 并非真正的电话拨号, 而是用户输入账号、密码, 通过身份验证, 获得一个动态的 IP 地址, 可以掌握上网的主动性】。

正确答案: D. 专线接入和虚拟拨号。

ADSL 是数字订户线路 (Digital Subscriber Line, DSL) 的一种。DSL 是 20 世纪 90 年代初开始开发的, 使用数字编码技术通过标准电话线把计算机连接到因特网的宽带通信技术。这种技术将电话线分成电话通道、上行通道和下行通道, 大幅度地提高了地区普通电话线路传输影视数据和其他数据的速度。DSL 使用信息包交换技术, 运行时不受电话系统的约束, 电话公司就可提供数字服务而又不锁定电话线路。DSL 技术有对称和非对称之分, 采用非对称数字订户线路 (ADSL) 的下行速度比上行速度快, 它适合用于浏览因特网和影视点播 (VOD); 采用对称数字订户线路 (SDSL) 的上行和下行速度相同。DSL 的数据传输速度很大程度上取决于电话公司和客户之间的距离, 见表 4-12 和表 4-13。DSL 通常指 xDSL, 其中的 x 表示在 DSL 技术基础上开发的各种 DSL 技术, 如 ADSL, HDSL, IDSL, RADSL 或 SDSL。

表 4-12 部分非对称 DSL 技术性能 (能与模拟电话共享线路)

DSL 类型	最大上行速度 (Mbps)	最大下行速度 (Mbps)	电缆对数	最大传送距离 (英尺/米)
ADSL	1.0	8	1	18000/5486
RADSL	1.0	7	1	25000/7620
G.Lite	512 (Kbps)	1.5	1	25000/7620
VDSL	1.6	13	1	5000/1524
	3.2	26	1	3000/914
	6.4	52	1	1000/304

表 4-13 部分对称 DSL 技术性能 (不能与模拟电话共享线路)

类 型	最大上行和 下行速度	电缆对数	最大传送距离 (英尺/米)
HDSL	668 Kbps	2	12000/ 3657
	1.544 Mbps (T1)	2	12000/ 3657
	2.048 Mbps (E1)	2 或 3	12000/ 3657
HDSL-2	1.544 Mbps (T1)	1	18000/5486
	2.048 Mbps (E1)	1	18000/5486
SDSL	1.5 Mbps	1	9000/2743
	784 Kbps	1	15000/4572
	208 Kbps	1	20000/6096
	160 Kbps	1	22700/6919
IDSL	144 Kbps	1	26000/7925

### 4.2.5 多媒体内容编辑语言

#### 【例 4-20】超文本的概念

超文本是一种信息管理技术, 其组织形式以\_\_\_\_作为基本单位。

A. 文本    B. 结点 (Node)    C. 链 (Link)    D. 环球网 (Web)

分析:

超文本 (Hypertext) 是包含指向其他文档或文档元素的链接的电子文档。超文本是用非线性结构组织的, 用户可以方便地浏览与它相关的内容, 如在有关奥运会的文章中, 用户可以通过链接直接转到历届奥运会举办城市或奥运金牌得主情况的介绍。该术语是 Ted Nelson 在 1965 年杜撰的, 他在计算机上处理文本文件时创造了一种把文本中遇到的相关文本组织在一起的方法, 让计算机能够响应人的思维以及能够方便地获取所需要的信息。

结点 (Node) 通常指: (1) 连接到网络并能与它通信的任何可寻址的计算机或设备。

(2) 在树形结构中, 两条或多条线汇合的点。

环球网 (Web) 是指分布在全世界所有 HTTP 服务机上相互链接的超文本文档的集合。万维网上的文档是用超文本标记语言 (HTML)、可扩展标记语言 (XML) 等语言编写的称为页面或 Web 网页的文档, 由统一资源地址 (URL) 指定要访问的文档所在的机器和路径, 并从执行超文本传送协议 (HTTP) 的服务器传到用户。嵌在万维网文档中的代码称为标签, 它们将文档中特定的单词和图像与相应的 URL 相连, 因此用户只需按键盘上的按键或单击鼠标就可以访问不同地理位置上的多媒体文档和程序。访问 Web 页面的用户还可以通过页面上的链接, 从执行文件传送协议 (FTP) 的站点下载文件, 或使用电子邮件向其他用户发送消息。万维网最初是在 1989 年由 Timothy Berners—Lee 为欧洲粒子物理实验室 (CERN) 开发的信息网。

正确答案: B. 结点 (node)。

## 4.3 思考练习题及答案

### 思考练习题

#### 1. 数据压缩算法应用

计算机磁盘上的文本文件要用\_\_\_\_算法压缩。

- A. 有损压缩算法                      B. 无损压缩算法

分析:

文本和程序都要求压缩后的重构数据与压缩前的数据完全相同。

#### 2. 压缩算法类型

霍夫曼 (Huffman) 编码是 (1), 算术编码是 (2), LZW 编码是 (3), 词典编码是 (4), 行程长度编码 (RLE) (5)。

- (1) A. 有损数据压缩算法      B. 无损数据压缩算法  
(2) A. 有损数据压缩算法      B. 无损数据压缩算法  
(3) A. 有损数据压缩算法      B. 无损数据压缩算法  
(4) A. 有损数据压缩算法      B. 无损数据压缩算法  
(5) A. 有损数据压缩算法      B. 无损数据压缩算法

分析:

霍夫曼编码法是根据给定数据集中各元素所出现的频率来压缩数据的一种统计压缩编码方法, 对出现次数越多的元素 (如字母), 其编码的位数就越少; 算术编码是给已知统计信息的符号分配代码的数据压缩方法, 它使用 0 和 1 之间的实数的间隔长度表示信息, 可获得接近于熵的平均码长; 词典编码是利用数据本身包含重复代码这种特性的压缩技术, 适用于开始时不知道编码数据的统计特性, 也不一定允许事先

知道它们的统计特性的场合；LZW 压缩算法是 Terry Welch 于 1984 年在 Lempel—Ziv 基础上提出的压缩算法，它使用压缩字符流中的重复数据串生成输出码流；行程长度编码（Run-Length Encoding, RLE）是利用连续数据单元有相同数值这一特点对数据进行编码的方法，对相同的数值只编码一次，同时计算出相同数值重复出现的次数，直接用它对自然图像编码时，编码效率不高，因为在自然图像中数值相同的连续像素比较少。

### 3. 声音数字化

5 分钟、双声道、22.05 KHz 采样、16 位量化的声音，经 5:1 压缩后，其数字声音的数据量约为\_\_\_\_\_。

- A. 5.168 MB      B. 5.047 MB      C. 26.460 MB      D. 26.082 MB

分析：

声音数字化后的数据量可按下式计算：

文件的字节数=采样时间(秒)×采样频率(样本/秒)×位数(位/样本)×声道数/8

本题中的声音在未压缩前的容量为，

$$(5 \times 60 \times 22050 \times 16 \times 2) / 8 / (1024 \times 1024) = 25.23 \text{ MB}$$

经 5:1 压缩后，数字声音的数据量为  $25.23/5 = 5.047 \text{ MB}$ 。

### 4. 图像技术颜色模型的应用

显示彩色图像用 (1)，打印彩色图像用 (2)，网页颜色用 (3)，

(1) A. RGB 相加混色模型      B. CMYK 相减混色模型      C. 可靠选色表

(2) A. RGB 相加混色模型      B. CMYK 相减混色模型      C. 可靠选色表

(3) RGB 相加混色模型      B. CMYK 相减混色模型      C. 可靠选色表

分析：

A. RGB 相加混色模型。RGB 相加混色模型 (Additive Color Model) 是组合红色光、绿色光和蓝色光产生颜色的模型，从理论上讲，任何一种颜色都可用三种基本颜色光按不同的比例混合得到。

B. CMYK 相减混色模型。CMYK 颜色模型 (Cyan-Magenta-Yellow-Black Ink, CMYK) 是由青色 (Cyan)、品红 (Magenta)、黄色 (Yellow) 和黑色构成的颜色模型，主要用在彩色印刷中。该模型与 CMY 颜色模型类似，但用单独的黑颜色分量产生黑色，而不是将 100%青色、100%品红和 100%黄色混合生成黑色。它是 32 (4×8) 位每像素的四通道彩色图像模型。

C. 可靠选色表。网页可靠选色表 (Websafe Palette) 是由 216 种颜色组成的表，用在超文本标记语言 (HTML)，层叠样式表 (CSS) 和嵌入到 Web 网页的图像上，目的是想让所有使用 8 位即 256 种颜色的显示器显示出一致的颜色。由于现在大多数用户的显示器都使用 16 位和 24 位颜色的显示方式，使用网页可靠选色表时仍然有疑问，于是又从中选择了 22 种颜色组成新的选色表，称为真正可靠选色表 (Reallysafe, Palette)，保证



在显示器的所有方式下和在 Web 浏览器上能够显示正确的颜色。(详见 <http://www.webreference.com/html/reference/color/>)

### 5. 图像压缩技术与文件格式

下列图像文件存储格式是目前最常见的格式。存储 256 色的图像时,对图像质量有损失的存储格式是\_\_\_\_\_。

- A. BMP      B. GIF      C. JPG      D. PNG

分析:

在这四种图像文件存储格式中,只有使用 JPG 文件格式存储时对视像质量才有损。

为加深对图像文件格式特性的了解,下面对这些图像格式做进一步的介绍。

(1) BMP: 使用 BMP 格式存储文件时,不对图像进行压缩。

(2) GIF: GIF 文件格式采用了 LZW (Lempel—Ziv Walch) 压缩算法对图像数据进行压缩,而 LZW 是无损数据压缩算法。

(3) JPG: JPEG 专家组开发了两种基本的压缩算法,一种是采用以离散余弦变换 (Discrete Cosine Transform, DCT) 为基础的有损压缩算法,另一种是采用以预测技术为基础的无损压缩算法。使用有损压缩算法时,在压缩比为 25:1 的情况下,压缩后还原得到的图像与原始图像相比较,非图像专家难于找出它们之间的区别,因此得到了广泛的应用。例如,在 V—CD 和 DVD—Video 电视图像压缩技术中,就使用 JPEG 的有损压缩算法来取消空间方向上的冗余数据。常用的 JPEG 算法是数据有损的压缩算法。为了在图像质量相同的前提下进一步提高压缩比,近年来 JPEG 专家组一直在完善 JPEG 2000 (简称 JP 2000) 的标准,这个标准采用了小波变换 (Wavelet) 算法。

JPEG 压缩算法利用了人的视觉系统的特性,使用量化和无损压缩编码相结合来去掉视觉的冗余信息和数据本身的冗余信息。JPEG 算法框图 4-9 所示,压缩编码大致分成三个步骤:① 使用正向离散余弦变换 (Forward Discrete Cosine Transform, FDCT) 把空间域表示的图变换成频率域表示的图。② 使用加权函数对 DCT 系数进行量化,这个加权函数对于人的视觉系统是最佳的。③ 使用赫夫曼可变字长编码器对量化系数进行编码。译码或者叫做解压缩的过程与压缩编码过程正好相反。

在 JPEG 算法的计算过程中,使重构图像质量降低的最主要的原因是量化计算,其他的计算都不会使图像质量下降。

(4) PNG: 易转换网络图形文件格式 (Portable Network Graphic Format, PNG) 是 20 世纪 90 年代中期开始开发的图像文件存储格式,其目的是企图替代 GIF 和 TIFF 文件格式,同时增加一些 GIF 文件格式所不具备的特性,以及不受 GIF 格式的法律限制。PNG 是位图文件存储格式,用来存储灰度图像时灰度图像的深度可多到 16 位,存储彩色图像时深度可多到 48 位。此外,还可存储多到 16 位的  $\alpha$  通道数据。PNG 使用从 LZ77 派生的无损数据压缩算法。PNG 的名称来源于非官方的 “PNG's Not GIF”。

如果要更深入和更全面地了解图像、声音和其他文件的存储格式，建议访问网站：  
<http://www.wotsit.org/>（2004—11—6 重访）。

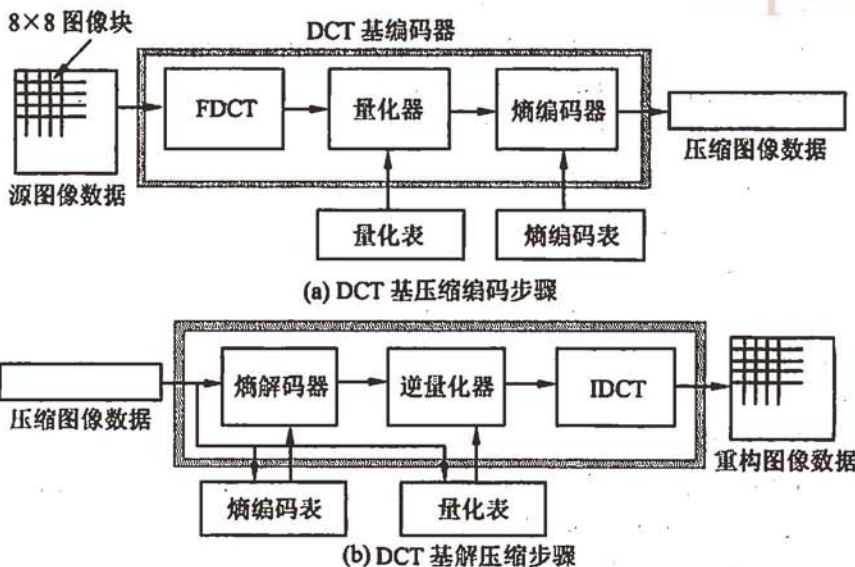


图 4-9 JPEG 压缩编码—解压缩算法框图

## 6. 数字电视图像压缩

对动态图像进行压缩处理的基本条件是：动态图像中帧与帧之间具有\_\_\_\_\_。

- A. 相关性      B. 无关性      C. 相似性      D. 相同性

## 7. MPEG—4

MPEG—4 是 (1)，MPEG—4 主要由声音编码、视像编码、数据平面、(2)、缓冲区管理和实施识别等部分构成。其中，数据平面包括 (3) 两部分。

- (1) A. 电视图像和伴音信息的通用编码  
 B. 高数据速率数字存储媒体的电视图像和伴音编码  
 C. 一套多媒体内容描述符的接口标准  
 D. 一套多媒体通信标准
- (2) A. 对象基表达    B. 场景描述    C. 合成编码    D. 描述符接口
- (3) A. 非可分等级编码模式和可分级编码模式  
 B. 合成数据对象和自然数据对象  
 C. 传输关系和媒体关系  
 D. 具有特殊品质服务 (QoS) 的信道和面向每个基本流的带宽

点评：

要正确理解试题的 (2) 和 (3)，需要认真研究 MPEG—4 标准文件，如 “International

Organisation for Standardisation Organisation / Internationale De Normalisation, Coding Of Moving Pictures And Audio, ISO/IEC JTC1/SC29/WG11 N4668, March 2002” (<http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm>), 查看是否有以及如何定义“数据平面(Data Plane)”的。此外,可阅读文献“Realizing MPEG-4 Streaming Over the Internet: A Client/Server Architecture using DMIF” (<http://lan.ece.ubc.ca/DMIF-ITCC2001.pdf>), 看看该文章是如何定义和解释“数据平面”的

### 8. MPEG 的压缩方法

VCD 的图像序列由帧内图像(I)、预测图像(P)和(1)构成。其中(2)采用 JPEG 压缩方法来去掉冗余信息。

- (1) A. 静止图像      B. 动态图像      C. 插补图像(B)      D. 电视图像  
(2) A. 帧内图像      B. 动态图像      C. 插补图像(B)      D. 电视图像

分析:

(1) VCD 上的影视图像是用 MPEG—1 方法压缩的。

本题中的“插补图像(B)”应该是指“双向预测图像(Bidirectional predictive-coded picture / Bidirectionally predictive-coded frame)”, 也称 B 图像。

视像压缩是利用每帧图像的空间冗余特性、帧图像之间的时间冗余特性以及人的视觉系统对颜色的感知特性, 采用数学方法用较少的数据表示原始数字化视像数据量的过程。B 图像是 MPEG 标准定义的三种帧图像之一, 以在它前后的帧图像作参考, 对它们的差值进行编码。在 MPEG 标准定义的 B 图像、I 图像和 P 图像中, B 图像是压缩率最大的图像。I 图像和 P 图像编码时, B 图像也不作为它们的参考图像, 因此 B 图像也不传播编码误差。

(2) 参见本章思考练习题答案。

### 9. 数字视像的普通常识

若电视图像每帧的数据量为 8.4 MB, 帧速率为 25 帧/秒, 则显示 10s 的视像信息, 其原始数据量为(1)MB。考虑存储和传输的要求, 可使用(2)标准对原始视像数据进行有效的压缩。

- (1) A. 84                  B. 840                  C. 2100                  D. 4200  
(2) A. JPEG              B. MPEG              C. MIDI                  D. Video

### 10. 多媒体存储装置 CD 盘物理格式

市场上出售一种标有“80 min, 730 MB”的 CD—R 光盘, 计算用于录制 80 分钟的 CD—DA 声音时, 这种 CD—R 光盘的容量为(1)MB。在计算机系统中, 使用 CD—ROM Mode 1 的格式来存放错码率小于 $10^{-12}$ 的“用户数据”, 用于录制这种数据时 CD—R 光盘的容量为(2)的 MB。由于录制声音数据、静态图像或电视图像数据时, 错码率可以降低要求, 可采用 CD—ROM mode 2 的格式, 用于录制这种数据时

CD—R 光盘的容量为 (3) MB。(要求写出计算式,  $1\text{MB}=1024\times 1024$  bytes。四舍五入取整数)。

(1) A. 730      B. 703      C. 808      D. 802      E. 650

(1) A. 730      B. 703      C. 808      D. 802      E. 650

(1) A. 730      B. 703      C. 808      D. 802      E. 650

分析:

正确回答这个问题的关键是要知道 CD—ROM 的地址结构。CD—ROM 的扇区地址与磁盘的扇区地址不同。磁盘的扇区地址是用 C—H—S (柱面号—磁头号—扇区号) 地址结构表示, 而 CD—ROM 是用计时系统中的分、秒, 以及特地为 CD—ROM 规定的分秒 (1/75 秒) 来表示。其次要知道 CD—ROM Mode 1 和 CD—ROM Mode 2 的每扇区的“用户数据”字节数。

(1) CD—DA

$80\text{分钟}\times 60\text{秒/分钟}\times 44\,100\text{样本/秒}\times 2\text{字节/样本}\times 2(\text{声道})/(1024\times 1024)$   
 $=846.720\times 10^6/(1024\times 1024)\text{字节}\approx 807.495\approx 808(\text{MB})$

(2) CD—ROM Mode 1 的用户数据: 2048 字节/扇区

$80\text{分钟}\times 60\text{秒/分钟}\times 75\text{扇区/秒}\times 2048\text{字节/扇区}/(1024\times 1024)$   
 $=737.28\times 10^6/(1024\times 1024)\text{字节}\approx 703.125(\text{MB})\approx 703(\text{MB})$

(3) CD—ROM Mode 2 用户数据: 2336 字节/扇区

$80\text{分钟}\times 60\text{秒/分钟}\times 75\text{扇区/秒}\times 2336\text{字节/扇区}/(1024\times 1024)$   
 $=840.96\times 10^6/(1024\times 1024)\text{字节}\approx 802.002(\text{MB})\approx 802(\text{MB})$

(4) CD—ROM 是从 74 分钟的 CD—DA 发展而来的。在计算机系统中, 使用 CD—ROM Mode 1 的格式来存放错码率小于  $10^{-12}$  的“用户数据”。一片 CD—ROM 光盘的存储容量为

$74\times 60\times 75\times 2048/(1024\times 1024)=650(\text{MB})$

通过这道练习题可知, ① 市场上销售的 CD—R 容量是什么含义, ② 从 (1) 和 (3) 的计算结果可知, CD—R 用于记录声音、图像和影视数据时的容量差不多。

### 11. VCD 盘上的影视

在 VCD (Video—CD) 光盘存储器上存储的是 MPEG—1 质量的影视节目, 标准 VCD 光盘的数据传输率最高可达到 1.4112 Mb/s, 分配给视像的数据传输率为 1.15 Mb/s, 使用 MPEG Layer 2 质量的声音, 它的数据传输率最高为 256 Kb/s, 使用市场上出售的标有“80 min, 730 MB”的 CD—R 光盘来录制 MPEG—1 的影视, 录制的影视节目最长为 \_\_\_\_\_ 分钟。

A. 80 分钟

B. 74 分钟

C. 60 分钟

分析:

$840.96\times 10^6/(1.15+0.256)/8\approx 4785(\text{秒})\approx 80\text{分钟}$



### 12. 多媒体网络应用：影视点播

在带宽为 10 Mbps 的局域网上各组点播不同内容的 MPEG-1 电视, 在满足播放要求的前提下可容纳的最大用户组数\_\_\_\_个。

A. 7

B. 8

C. 9

分析:

MPEG—1 的数据传输率约为 1.41 Mb/s, 分配给视像的数据传输率为 1.15 Mb/s, 使用 MPEG Layer 2 质量的声音, 它的数据传输率最高为 256 Kb/s, 因此, 可容纳的最大用户数组数为:  $10/1.41 \approx 7$

\*影视点播 (Video on Demand, VOD) 是在网上提供的允许用户自己选择影视节目的影视服务。其播放控制和操作方式与录像机类似, 如播放、暂停、快进、快退等多种功能。每当用户请求观看影视时, VOD 系统就把影视节目传送给用户的接收和显示装置, 用户可不必购买录像带、VCD 或 DVD 盘, 节目集中存放在影视库中。VOD 系统通常采用客户机/服务机模式, 服务机端的主要配置包括计算机系统、影视服务程序 (或叫做影视服务器) 和影视库, 客户端的配置有以下几种形式供选择: ①PC+播放器; ②机顶盒+电视机; ③WebTV 盒+电视机。

### 13. 多媒体内容编辑语言

由于这部分内容的试题不多, 而且主要是靠编程实践才能掌握, 因此有兴趣的读者可做《多媒体技术基础》第 2 版教材第 21 章“超文本标记语言 (HTML)”和第 22 章“可扩展标记语言 (XML)”中的“练习与思考题”, 在教材上也附有答案, 在此不再引用。

### 思考练习题答案

1. B

2. (1) B (2) B (3) B (4) B (5) B

3. B

4. (1) A (2) B (3) C

5. C

6. A

7. (1) D (2) B (3) C

8. (1) C (2) A

9. (1) C (2) B

10. (1) C (2) B (3) D

11. A

12. A

## 第5章 网络基础知识

### 5.1 内容提要

本章内容主要包括了计算机网络的基本原理、实际应用和网络安全等。基本原理部分包括计算机网络的功能和组成,网络的层次体系结构,构建网络的常用设备和不同网络采用的网络协议原理;实际应用部分讨论了 Internet 常见应用的基本工作过程和常见的 Windows NT 网络的配置管理。最后讨论的是网络安全问题,包括信息安全和路由安全(防火墙技术)两个方面。整个章节从网络的软硬件组成和应用角度讲述了网络基础知识。本章内容组成上可以分成4部分内容:网络的基本概念、构建网络的基本设备和协议、Internet 应用原理和 Windows NT 应用管理、网络安全基础。

#### 5.1.1 计算机网络的基本概念

##### 内容要点

##### (1) 计算机网络的定义

根据计算机网络的不同发展历史,计算机网络定义的内涵也在变化,现在,人们普遍接收的定义是:地理上分布的、具有自治能力的计算机通过传输媒体连接起来,遵循一定的通信协议进行相互通信和资源共享。

##### (2) 计算机网络的基本组成

计算机网络的组成可以从物理上和逻辑上的组成看待,物理上的组成根据不同的组网方式有所差别。从逻辑上看来,计算机网络由通信子网和资源子网组成,通信子网负责网络信息的传输,资源子网负责信息的处理。因此,通信子网是指包括了通信线路、通信交换设备、计算机中的网卡等主要进行信息传输的设备的总称。而资源子网是指连接在网络上的进行信息处理的计算机及终端设备的总称。

##### (3) 网络从不同角度分类方法

可以从不同的角度来分类:常用的分类方法是从网络的地理覆盖范围分,分为局域网、城域网、广域网,有的将广域网进一步分出互联网。

##### (4) 网络的拓扑结构

可以通过不同的线路连接方式将计算机连接成网络,这种连接各计算机的物理形状称为拓扑结构,拓扑结构有:星型、环型、树型、总线型、不规则的网状(分布式)连接。不同的结构有不同的优点和特性。

### 5.1.2 计算机网络的体系结构

#### 内容要点

##### (1) 计算机网络体系结构的概念

所谓网络的体系结构主要是指构成网络系统的软硬件作为整个一个系统, 考虑其功能如何进行分配, 如何将异构的计算机 (不同的数据表示、不同的操作系统、不同的应用等) 相互连接, 完成计算机网络的功能等问题。网络体系结构主要是 ISO 提出的 OSI/RM (开放系统互连参考模型), 定义了 7 个层次的协议功能。

##### (2) 网络协议的定义

协议是指为了能够相互通信, 参加通信的计算机必须遵循的一组通信规则或约定。

##### (3) 网络协议的层次体系模型

ISO 定义的 7 层协议参考模型 OSI/RM, 将网络的体系结构从低到高划分为 7 个层次, 分别是物理层、数据链路层、网络层、运输层、会话层、表示层和应用层。由于该体系结构的复杂性, 没有得到业界的很好支持, 另外一个事实标准的体系结构 TCP/IP 协议簇反倒得到很好的应用, TCP/IP 是一个 5 层的体系结构, 在主要的层次体系结构中 OSI/RM 基本一致。

##### (4) 层次结构中, 数据的流向和工作原理

网络体系结构中, 每层协议都具有不同的功能, 在一个系统中 (一台计算机中) 相邻的不同层次间的数据进行上下传递时, 下面的每一层次都要将上层传递来的信息加上自己层次的协议控制信息, 以便对方对等层次 (Peer) 识别, 达到与对等层相互通信的目的。在同一系统中, 下层将上层传来的信息当做数据对待, 传到对方后, 下层去掉对等层的控制信息, 将数据传到上层。在整个过程中, 各层协议通过控制信息完成与对等层的协议功能。

### 5.1.3 网络的传输控制

#### 内容要点

##### (1) 常用的传输介质

传输介质分为有线介质和无线介质。常用的有线介质有: 双绞线、同轴电缆和光纤等; 无线介质有: 微波、红外线和卫星等。

##### (2) 信道复用技术

根据信道上信号的表现形式 (数字信号、模拟信号), 复用方式也不同。频分复用 FDM: 是把线路或空间的频带资源分成多个频段 (带), 将其分别分配给多个用户, 每个用户终端的数据通过分配给它的子通路 (频段) 传输, 频分复用适合于模拟信号的传输。时分多路复用 TDM: 是将传输信号的时间进行分割, 使不同的信号在不同时间内传送, 即将整个传输时间分为许多时间间隔 (称为时隙、时间片等, Slot Time)。每个时间

片被一路信号占用。换句话说, TDM 就是通过在时间上交叉发送每一路信号的一部分来实现一条线路传送多路信号。波分复用 WDM (Wavelength Division Multiplexing): 在光纤信道上处于不同波段的两个光合成共享一根光纤, 传送到远方的目的地, 随后再将它们分解开来。

### (3) 同步、异步传输

在异步传输方式中, 每次传送一个字符 (5~8 位), 都在每个字符代码前加一起始位, 表示该字符代码的开始。在字符和校验码后加一停止位, 以示该字符的结束。所以又称起止式同步, 典型的传输方式是 RS-232C 的传输。

同步传输方式中, 利用时钟的同步使发送和接收装置之间的定时不发生误差。同步传输一般是多个二进制位组成数据块, 数据块前、后用加上同步定界符等控制信息组成“帧”。同步传输分为面向比特 (位) 和面向字符的方式。

### (4) 检错纠错方式

为防止传输过程的错误, 需要进行检错和纠错。为检查错误而对传输信息进行的编码称为检错码。常见的检错码有奇偶校验码和循环冗余码 (CRC), 能够根据编码纠正错误的称为纠错码, 如海明码。

### (5) 数据交换方式

网络中的数据交换方式指数据怎样通过交换结点在网络上传输, 交换方式分为线路交换、报文交换、分组交换、信元交换等几种方式。

## 5.1.4 网络互连设备

### 内容要点

#### (1) 网络互连的层次与设备

网络互连时, 一般要通过中间设备相连, 这些设备主要有: 中继器 (又称转发器, 在物理层实现互连)、网桥 (又称桥接器, 在数据链路层实现互连)、路由器 (在网络层实现互连)、网关 (又称网间连接器或连网机, 在传输层及以上的层次实现互连) 等。网络之间通过互连设备可以将不同实现技术的单个网络连接形成更大范围的网, 互连在哪一级上实现, 取决于实际需要和网络之间的兼容程度。

#### (2) 构建局域网的组成部分

构建局域网的设备主要包括计算机、网卡、交换机、传输媒体和路由器等网络互连设备。根据构建网络的范围不同, 在网络中使用的传输媒体、网络的拓扑结构、传输速度以及选取的不同档次的网络设备也不同。

#### (3) 接入 Internet 的方式

接入 Internet 分为单台 PC 机 (家庭) 接入和整个网络 (单位) 接入两种。提供给单台 PC 机接入技术包括 POTS (普通电话服务) 拨号、ISDN (窄带的综合业务数字网)、ADSL (非同步数字用户环路)、Cable Modem (电缆调制解调器) 等接入。面向大用户



的有公共分组交换网接入、帧中继接入和光纤接入等。

### 5.1.5 局域网技术

#### 内容要点

##### (1) 局域网模型

如图 5-1，局域网模型主要规定了低 2 层的标准。

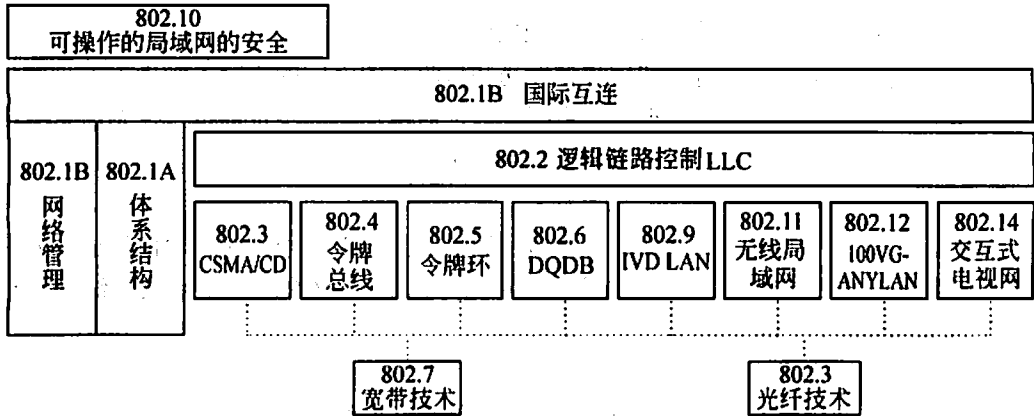


图 5-1 IEEE 802 标准系列之间的关系

##### (2) 以太网技术

以太网采用 CSMA/CD 访问控制方式，二进制指数退避算法。以太网分标准以太网（802.3）、快速以太网（100M）（802.3u）和千兆以太网（802.3z）。快速以太网和千兆以太网联网的关键设备是以太网交换机。

##### (3) 令牌环（Token Ring）技术

采用令牌沿环传递来分配带宽方式，令牌环拓扑结构是环型的，编码方法为差分曼彻斯特编码。标准是 IEEE802.5。

##### (4) 光纤分布式数据接口（Fiber Distributed Data Interface, FDDI）

FDDI 技术标准规定了一个 100Mbps 光纤环型局域网的介质访问控制协议和物理层规范。访问控制方式与 802.5 相似，但具体实现上有所区别，并不像 802.5 那样要等待自己发出的帧收到后再释放令牌。FDDI 采用一种新的编码技术，称为 4B/5B 编码。

### 5.1.6 广域网与接入技术

#### 内容要点

##### (1) PPP 协议

PPP 协议是用于拨号上网和路由器之间连接的点到点通信协议，属于数据链路层的

一个协议,它具有进行错误检测、支持多种协议、连接时允许商议 IP 地址、允许身份验证等功能。

### (2) DDN

DDN 即数据数字网。以专线形式提供给用户固定出租线路。DDN 网络中间没有交换,网络传输速率高、时延小、质量好。

### (3) ISDN

综合业务数字网 (Integrated Services Digital Network),主要的通信业务的功能是进行音频和数字化的语音传输,以及提供了 64Kbps 的电路交换数字信道、分组交换虚拟电路和无连接业务。ISDN 标准指定了基本速率接口 (BRI) 和主速率接口 (PRI)。基本访问速率是 2B+D,主速率接口在欧洲是 30B+D,而在美国,日本,加拿大是 23B+D。我国采用 30B+D 接口。早期的 ISDN 称为 N-ISDN (窄带 ISDN),现在以 ATM 为基础的称为 B-ISDN (宽带 ISDN)。

### (4) FR (Frame Relay) 帧中继

帧中继是在 X.25 基础上发展起来的业务。在 X.25 的制定中,由于底层通信线路传输不可靠,所以运行 X.25 的中间结点要花大量的时间进行分组的错误检验和纠错,当通信线路采用光纤技术后,由于数据传输基本不出错,就可以简化 X.25 的许多功能,极大地提高了数据传输过程中的交换速度。帧中继在虚拟电路 (在帧中继上,称为虚拟连接) 上使用帧交换技术,并有交换型 (SVC) 和永久型 (PVC) 两种。

### (5) 异步传输模式 (ATM) 技术

ATM 是在分组交换技术上发展起来的一种快速分组交换,它吸取了分组交换高效率 and 电路交换高速率的优点,并且和分组交换、帧中继一样都可以实现对网络资源的按需分配。这种交换称为信元交换。ATM 功能层次上划分为物理层、ATM 层、ATM 适配层 (AAL) 和高层。ATM 技术虽然可以提供高速可靠的数据传输,但因为与传统的以太网技术不能很好地兼容,需要进行局网仿真。

## 5.1.7 TCP/IP 与 Internet

### 内容要点

#### (1) TCP/IP 协议簇

TCP/IP 协议簇是 Internet 网络使用的基本协议,这个协议簇中,以 TCP 和 IP 为主要的协议,称为 TCP/IP 协议。整个协议簇中的结构如图 5-2。

这个协议簇中,底层是采用不同技术的网络,它们完成 IP 报文的传输,统称为网络接口层。上面是 IP 协议和其他几个辅助协议。之上是面向连接的 TCP 协议和无连接的 UDP 协议。最上是面向不同应用的应用层协议。

#### (2) IP 协议工作过程

IP 协议有两种版本成为标准,它们分别是 IPv4 和 IPv6,IP 协议完成的主要功能是

控制 IP 报文在网络上的传输, 为此 IP 协议要完成: ① 寻址 (路由选择), 根据 IP 报头中的目的 IP 地址和路由向前转发 IP 包。② 分段/重组, 当下层网络能传输的最大数据包长度小于当前的 IP 包时, 需要将 IP 包分段, 在目的结点需要将分段的 IP 包重组。

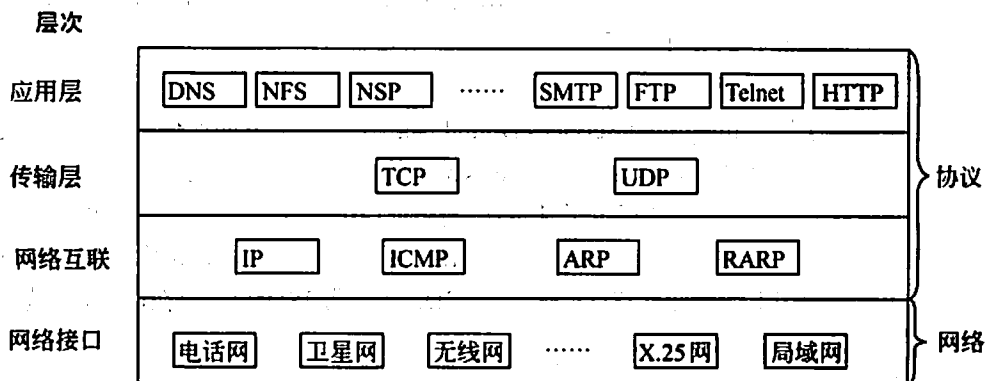


图 5-2 TCP/IP 协议簇及层次关系

### (3) TCP 协议工作过程

运输层主要有两个协议: TCP 协议 (Transmission Control Protocol) 和 UDP 协议 (User Datagram Protocol)。它们下层都使用 IP 协议。UDP 在传送数据之前, 不需要建立连接, 同时, 远程主机收到 UDP 数据包后, 不需要返回任何响应, 属于不可靠的传输。TCP 则提供面向连接的可靠传输服务。

TCP 与 UDP 通过端口 (port) 为上层的应用进程提供服务。端口是一个 16 bit 的地址, 用端口号进行标识。端口号分为两类。一类是专门分配给一些最常用的应用层程序, 被称为熟知端口 (Well-Known Port), 数据范围为 0~255。另一类是一般的端口号, 用来随时分配给请求通信的客户进程。

TCP 提供面向连接的、可靠的、全双工的、点对点通信服务。它可以保证数据按照顺序, 没有重复可靠的送达。TCP 要解决包的丢失与重发、拥塞控制等问题, TCP 使用三次握手方式来确保连接的建立和终止都是可靠的。

### (4) IP 地址的概念

IP 地址是识别在 Internet 上的计算机的一个地址, 在整个 Internet 上 IP 地址必须惟一。数据包在网络上传输时, 根据它包含的目的计算机的 IP 地址将数据包送达目的地。IPv4 的地址是 32 比特, 即 4 个字节, 每个字节用十进制表示, 字节之间用 “.” 分隔。IPv4 的地址划分为 5 类, 常用的是 3 类, 即 A B C 三类, IP 地址是由网络号和主机号两部分组成的, 在 Internet 传播时, 路由器是根据 IP 地址的网络号来进行路由选择的。

IPv6 中, 地址被扩充为 128 位, 解决了有限的地址数量问题, IPv6 的报头设计克服了引进扩展和可选项机制的困难以及差的安全性和隐私性问题。

### (5) 域名服务 DNS

DNS (Domain Name System) 是解决 IP 地址与计算机名字的对应问题。由于数字表示的 IP 地址比较难记, 出现了用字母表示的计算机的名字, 称之为域名。例如 mail.cs.tsinghua.edu.cn 表示清华大学计算机系一台叫做 mail 的计算机, 用这种方式来表示 Internet 的计算机地址就容易记住, 为了将域名转换成 IP 地址使得数据报可以根据 IP 地址在网络上传输, 就出现了 DNS 服务, 它的功能就是自动将域名转换成 IP 地址。DNS 系统中名字是按层次结构组织的, 按非集中式管理, 目录服务由分布式 DNS 服务器提供。

### (6) 其他 Internet 应用

其他常用的 Internet 应用层协议有文件传输协议 FTP、远程登录协议 Telnet、电子邮件系统 SMTP (Simple Mail Transfer Protocol) 协议、WWW 使用的超文本传送协议 HTTP 等。

## 5.1.8 客户机/服务器模式与网络计算

### 内容要点

#### (1) 文件服务器模式及域模式

它是局域网中使用的用于网络管理和资源共享的模式。包括文件服务器为主体的使用方式, 如 Novell Netware 网络操作系统中, 文件服务器提供给各网络工作站共享的大磁盘空间。工作组为主体的方式, 如 Microsoft Windows 3.11 for workgroup, LAN 上的在同一个工作组上的计算机可以以对等的方式访问别人或被别人访问。以域模型为主体的使用方式, 如 Windows NT (2000) Server 等, 有一个域服务器对网上加入该域的用户进行安全和访问资源的控制, 因此只要用户通过了域服务器的检验, 就可以以一定的权限访问全域上的资源。

#### (2) 客户机/服务器 (C/S) 模式

在局域网中, 客户机发出请求, 服务器完成数据计算并返回结果的一种计算模式, 可以细分为 3 层模式: 客户机—应用服务器—数据库服务器。

#### (3) 以 Internet/Intranet 为网络环境的 B/S (Browser/Server) 模式

将 WEB 技术和数据库结合在一起, 形成跨平台开放性的具有多媒体应用特征的信息服务。包括的技术有 TCP/IP、数据库、Web 技术和 HTTP 协议等。

## 5.1.9 Windows NT 系统及管理

### 内容要点

#### (1) Windows NT 开放的体系结构与网络协议

网络结构是按层次组织的, 并提供了扩充能力。对下提供了 NDIS (网络设备接口规范), 对上提供了 TDI (传输驱动程序接口)。主要网络协议为 TCP/IP 协议, 常用的还有 NetBEUI 协议。



### (2) 分布式处理的多进程间通信 (IPC) 机制

IPC 包括了命名管道和消息通信机制、NetBIOS 通信方式、Windows Sockets 通信机制、远程过程调用 (RPC) 机制和网络动态数据交换 (NetDDE)。

### (3) Windows NT 网络模型

两种模型的网络：域模型和工作组模型。其中域模型包括单域模型、主域模型、多主域模型和完全信任模型等 4 种。

## 5.1.10 网络安全

### 内容要点

#### (1) 网络安全的内容和安全机制

网络安全包括信息存储安全和路由信道安全两个方面，信息存储安全主要是保证信息不被非法地获取、篡改和病毒的感染等。路由安全是指在信息的传输过程中，不被获取、攻击和改变传输路径等。采取的安全机制主要包括：用户身份的确认、访问权限限制、信息加密以及信息的完整性判断等。信息加密机制主要分为数据的存储加密和数据的传输加密，加密机制是采用密码技术来实施的。

#### (2) 信息存储安全技术

包括资源的访问控制、身份验证与数据加密技术，涉及到数据加密算法和身份验证的安全协议和技术。

#### (3) 路由信道安全技术

包括传输信息的数据加密和防止攻击的防火墙技术。防火墙技术分成两大类：网络级防火墙和应用级防火墙。

## 5.2 例题分析

**【例 5-1】**在现代的网络组网中，一般都以双绞线为最终连接桌面的网线，接交换机、HUB。路由器等网络交换或路由设备。从工作方式看来，当计算机以交换机为中心连接时，网络的拓扑结构是(1)，当交换机以共享式 HUB 连接时，它们形成的实际的拓扑结构(2)，当一个网络中存在多个路由器时，这时的计算机间的连接方式是(3)。

可选答案：

(1) ~ (3)：A. 星型      B. 总线      C. 树型      D. 不规则网状型

正确答案：(1) A    (2) B    (3) D

分析：

计算机网络的拓扑结构分为总线型、星型、环型、树型、分布式结构等，一般局域网的拓扑结构是总线型、星型、环型、树型，广域网的拓扑结构是不规则的网状结构（分布式）。最初，标准的以太网（10BASE-5）是总线型的，随着网络技术的发展，出

现了连接双绞线的 HUB 和交换机,在以 CSMA/CD 工作方式的以太网中引入了交换技术,双绞线连接方式从物理上看来是星型结构连接的,当从各计算机共享线路的工作方式看来,用共享 HUB 连接的计算机,各计算机之间在 HUB 中仍以总线方式共享信道,而交换机连接各计算机时,交换机的各个端口才是同时工作的,因此,从工作方式看来,共享 HUB 连接的各计算机仍是总线共享式的,而交换机连接的是星型结构的。

采用环型结构的局域网有 Token Ring、FDDI 网络。

采用星型结构的网络除交换式以太网外,常见的还有 ATM 局域网。

当网络中有多个路由器连接时,可以将路由器连接的线路看成是广域网连接,由于路由器之间连接的不规则性(任意两个之间可以连接或不连接),因此这种连接方式可以看成是不规则的分布式结构。

【例 5-2】 局域网的基本特征可以从以下几个方面来描述,地理上覆盖范围一般在(1),传输速率属于(2),所有权归(3),符合(4)制定的标准,对应 OSI 模型的(5)。

可选答案:

- (1) A. 几米以内      B. 几公里以内      C. 几十公里以内      D. 几百公里以内  
(2) A. 56KBPS 以内      B. 10M 以内      C. 1G 以内      D. 10G 以上  
(3) A. 公用      B. 个人      C. 全球      D. 一个机构  
(4) A. IEEE      B. ITU      C. IAB      D. EIA  
(5) A. 物理层      B. 数据链路层      C. 物理和数据链路层      D. 数据链路和网络层

正确答案:

- (1) B    (2) C    (3) D    (4) A    (5) C

分析:

局域网是从网络的物理覆盖范围角度来分类的一种网络,一般为一个单位或一个机构所有,在一个校园、一个工厂或一栋大楼内铺设,因此范围不超过几公里。通常它分布在一个学校或一个企业单位,为本单位使用。一般称为“园区网”或“校园网”。

开始的局域网传输速率从几兆到 100M,以后发展成 1000 兆,即 1G。

局域网还有的一些特征是:数据传输可靠,误码率低。误码率一般为  $10^{-8} \sim 10^{-11}$ 。拓扑结构简单,大多采用总线、星型、环型等。

与网络标准有关的几个主要组织有 IEEE、ISO、ITU、IAB 和 EIA 等。

局域网的标准主要遵循 IEEE 制定的 802 标准,也被 ISO 采纳为 8802 标准,ITU 是国际电信联盟,著名的 X 系列标准就是由已改组成为 ITU 下属的 CCITT 制定的。IAB (Internet Architecture Board) 是 Internet 体系结构局,其下属的 IEIF (Internet 工程特别任务组) 致力于 Internet 近期发展的工程问题。EIA 是美国电子工业协会,其著名的标准有 RS-232 以及 RS-449 等。

IEEE802 标准主要制定的是网络的低 2 层的标准,对应 ISO 提出的 7 层网络协议的

物理层和数据链路层,如图 5-3 所示:其中局域网标准将数据链路层分为逻辑链路控制层(LLC)和介质访问控制层(MAC)。MAC 层和物理层一起构成适合不同的网络传输控制的标准,如 803.3、802.4、802.5 等。

802.1 B 网际互连							
8 0 2  1 B	8 0 2  1 A	802.2. LLC					数据链 路层
		802.3 CSMA/CD	802.4 Token Bus	802.5 Token Ring	802.6 MAN	其他	
		802.3	802.4	802.5	802.6		物理层

图 5-3 IEEE 802 标准的 MAC 层和物理层

**【例 5-3】** 网络的拓扑结构是指连结成网的物理形状,拓扑结构可以分成(1)方式和(2)方式,(1)方式有(3)、树型、回路形(包括相交回路型)、全连接网状和(4)。(2)方式包括总线型、环型和(5)。

可选答案:

- (1). (2)    A. 点对点            B. 广播            C. 有线            D. 无线  
 (3)            A. 星型            B. 环型            C. 无线            D. 总线  
 (4). (5)    A. 星型            B. 不规则连接    C. 无线            D. 分布型

答案:

- (1) A    (2) B    (3) A    (4) B    (5) C

分析:

网络的拓扑结构细分为许多方式,但一般地讲,通信子网可以设计成 2 种通信(信道)类型:点对点通信(Point-to-Point)和广播通信(Broadcast)。

点对点信道其特点是一条线路连接一对结点。两台主机常常经过几个结点相连接。信息的传输采用存储转发方式。这种信道形成的通信子网常见的拓扑结构有:①星形,②树形,③回路形,④相交回路形,⑤全连接形,⑥不规则形式分布式。

广播信道其特点是只有一条供诸结点共享的通信信道。任一结点所发出的信息报文可被所有其他结点接收,当然,对信道需要有一定的访问控制机制。由这种信道构成的通信子网的拓扑结构可有三种形式:①总线性,②环形,③卫星或无线广播通信方式。

**【例 5-4】** 国际标准化组织的 OSI 基本参考模型共有(1)层。HDLC 和 TCP、IP、UDP 分别是对应于该模型中(2)、(3)、(4)和(5)协议。

可选答案

- (1):            A. 5            B. 6            C. 7            D. 8  
 (2) ~ (5): A. 数据链路层    B. 网络层    C. 运输层    D. 应用层



答案:

(1) C (2) A (3) C (4) B (5) C

分析:

OSI 参考模型将网络分为 7 层, 从低到高分别如下所述。

(1) 物理层: 规定在物理媒体上传输信号时双方接口的机械特性、信号的电气特性、使用信号的连线的功能特性和控制时序的规程特性。实现物理层协议的典型实例是 RS-232C。

(2) 数据链路层: 通过数据链路层协议, 将不够可靠的传输信道变成可靠传输“数据帧”的数据链路。数据链路层协议是通过对接收到的数据帧的检错和丢弃重来保证数据在链路上进行正确传送的, 数据链路上的数据单元称为“帧”, 数据链路层的典型例子是高级数据链路控制规程 HDLC, TCP/IP 协议中, 数据链路层也被称为网络接口层。

(3) 网络层: 在网络层传输的数据格式叫做“数据分组”, 网络层的主要功能是进行路由选择和拥塞控制, 使数据分组根据目的地址或收发双方的连接从一个网络传送到另一个网络, 最终到达目的网络。网络层对高层提供两种传送服务方式: 数据报服务和虚电路服务。数据报方式也称无连接方式, 分组在发送时不需要与对方建立连接, 每个分组携带有目的网络地址, 在网络上独立的选择路由。虚电路方式也称为面向连接方式, 在数据发送之前, 收发双方要先建立一条连接, 叫做“虚电路”, 在一条虚电路传送的分组顺序是按序进行的, 分组只需携带虚电路号即可。传输结束后释放连接。

TCP/IP 协议中, 网络层也称为互联网层, 即 IP 层。

(4) 运输层: 运输层功能与数据链路层相似, 即进行差错控制、恢复处理和流量控制等, 但这层协议是在端到端的主机之间进行的, 而数据链路层协议是在点到点的通信子网内进行的。TCP 协议是运输层的一个主要的具体协议。TCP 提供面向连接的服务, TCP 上的连接是通过使用双方的一对套接字 (SOCKETS) 来标志的, 不同的套接字表示了不同的应用进程, 如 FTP 用的是 21, HTTP 用的是 80 等。

在高层协议的划分上, ISO/OSI 参考模型将其划分为 3 层, 即

(5) 会话层: 对不同主机的会话进程进行控制和管理。会话管理分为 3 个阶段, ① 会话建立, 会话层在会话建立时协商会话方式, 如单工方式或双工方式。② 会话管理, 进行同步控制, 在数据流中适当的位置插入同步点, 当传输出现中断时, 可以从同步点的位置开始重新传输, 避免了从头重传。③ 会话释放, 结束会话。

(6) 表示层: 为参加通信的有不同数据表示方式的计算机提供语法 (数据格式) 和语义 (信息的意思) 的变换。通信的计算机在数据表示上可能是异构的, 为了通信双方相互理解对方传来的数据, 需要网络来进行数据的转换, 做到异构网络的互连, 转换方法通常采用一种抽象的表示方法, 将各种不同的数据表示转换成这种中间的表示语法, 并通过传送语法传到接收方, 转换成接收方可识别的另一种数据表示。数据的加密、压缩也属表示层的范围。



(7) 应用层: 面向用户, 提供不同的使用手段和工具, 如电子邮件、浏览器、FTP、Telnet 等。

HDLIC 协议是数据链路层协议中一个非常典型的协议, 该协议从属于面向比特的同步传输协议, 相应的还有面向字符的同步传输协议, 如 IBM 的 BSC 协议 (二进制同步通信)。

与 ISO/OSI 的参考模型对应的另一个在 Internet 上使用的协议模型是 TCP/IP 协议, ISO/OSI 的参考模型和 TCP/IP 协议在低 4 层上是一致的, 虽然有时叫法不一样。在高层上, TCP/IP 协议簇中, 将会话层、表示层和应用层合并为一层, 通称为应用层。将物理层和数据链路层看成是网络接口层。TCP 和 IP 分别是整个协议簇中的运输层和网络层, 因为这两个协议是整个协议簇中的主要协议, 将整个协议簇称为 TCP/IP 协议。

与 TCP 同属一层的协议还有 UDP 协议。TCP 是面向连接的协议, 叫做传输控制协议, 提供可靠的数据传送。而 UDP 是无连接的协议, 称为用户数据报协议。

**【例 5-5】** 用户在家里利用普通电话线上网, 他需在计算机上加接 (1), 该设备的作用是将计算机内的 B 转化成适于在电话线上传输的 C, 它的主要性能指标之一是传输数据的速率。目前常用来电话拨号上网的 A 的数据传输率可达 D bps, 其中 bps 表示每秒位数。假设在一分钟内需要传输 3600 个汉字 (双字节), 每传输一个字节需要另加二位辅助位, 则所用的 A 至少应有 E bps 的数据传输速率。

可选答案

(1): A. 网卡            B. 声卡            C. MODEM            D. CDROM

(2)、(3): A. 视频信号            B. 脉冲编码信号  
          C. 已调制的音频信号        D. 数字编码信号

(4): A. 9600            B. 56K            C. 512K            D. 10M

(5): A. 1200            B. 9600            C. 28.8K            D. 56K

正确答案: (1) C    (2) D    (3) C    (4) B    (5) A

分析:

利用普通电话线上网时要用调制解调器, 传统的家庭上网是电话拨号方式, 现在使用 ADSL 接入, ADSL 中, 数据信号并不通过电话交换设备, 不需要拨号。无论哪种上网方式都需要调制解调器, 它的作用是将计算机的数字信号调制成适合于在电话线上传输的模拟信号, 即已经调制的音频信号。在另一头将模拟信号再解调成数字信号交给计算机。

调制技术就是使载波信号的幅度、频率或相位 (其中的一种或几种) 随发送信号变化的过程。这三种基本技术分别叫做幅移键控法 (ASK)、频移键控法 (FSK) 和相移键控法 (PSK), 载波信号是利用信号发生器产生的高频正弦波。发送信号 (可以是模拟数据或数字数据) 一经调制, 就将作为模拟信号 (已携带发送信号的载波——已调波), 通过送信介质发送出去。

调制解调器与计算机的最初制定的接口是 RS-232C 标准, 是美国电子工业协会 EIA 制订的物理层的一个通信标准, 它是串行传输的, 通常称为串口。它规定了通信双方的机械特性、功能特性、电气特性和规程特性。用电平+5V~+15V 表示“0”, 用-5V~-15V 表示“1”, 信号的传输速率在 20Kbps, 电线长度限于 15m 以内。

而目前市场上推出的新的拨号上网的调制解调器最高速率可达 56K。接口方式也可以有 USB 等方式。在传输汉字时, 因为一个汉字在计算机内部(汉字内码)占两个字节, 即 16 比特, 再加上 4 位辅助位, 传输一个汉字要传送 20 比特, 60 秒中要传输 3600 个汉字, 即  $3600 \times 20$  比特, 至少需要  $3600 \times 20 / 60 = 1200\text{bps}$ 。

**【例 5-6】** 某实验室要建立一个 20 台微机组成的局域网, 从节约费用的角度来看, 宜采用最通用的(1), 采用交换式 HUB 和双绞线进行连接, 使用的网络拓扑结构是(2), HUB 与微机工作站之间的最长距离为(3)米。

- (1) A. 以太网    B. 令牌环网    C. 令牌总线网    D. 双总线网  
(2) A. 总线型    B. 星型    C. 环型    D. 混合型  
(3) A. 100    B. 200    C. 400    D. 500

答案:

- (1) A    (2) B    (3) A

分析:

目前, 市面上流行的组网技术是采用以太网技术, 无论从技术的成熟度还是从网络设备的选择上, 以太网的被支持程度和成本都远远超过采用其他技术的网络, 而且, 可以根据不同的档次选择不同的设备类型, 以太网设备目前可以支持从最普通的 10M, 到 100M 和 1000M, 交换设备可以选择普通的 HUB、10M、100M、1000M 交换机, 电缆可以采用双绞线(较早使用的是同轴电缆)、光纤和无线通信。

令牌环(IEEE802.5)网络和令牌总线网(IEEE802.4)是在局域网发展的初期出现过的产品, 但得到很少生产厂家的支持。双总线(DQDB)更是如此。

用交换 HUB 和双绞线将各计算机连接起来, 从网络的物理形状上形成了交换 HUB 为中心的星型结构。如果 HUB 是交换式的, 则从信息的传播方式上看也是星型的, 但如果是共享式 HUB, 则在 HUB 内部信息的传播还是总线式以广播方式传送。

在使用双绞线作为传输介质时, 微机和 HUB(或交换机)之间的最大传输距离规定为 100 米。

**【例 5-7】** 网络按交换方式分为电路交换和(1), 其中(1)交换又分为(2)和(3), 而(3)又细分为(4)和(5), (4)方式又称面向连接方式, (5)方式又称无连接方式。

可选答案:

- (1)~(3) A. 分组交换    B. 报文交换    C. 数据报交换    D. 存储转发交换  
(4)、(5) A. 报文交换    B. 数据报交换    C. 虚电路交换    D. 存储转发交换

正确答案: (1) D    (2) B    (3) A    (4) C    (5) B

**分析:**

网络的信息交换方式最初是与电话通信一样的电路交换方式, 60 年代人们首次提出“分组”(Packet)一词。1969 年 12 月美国的分组交换网 ARPANET 投入运行。从此, 计算机网络的发展就进入了一个新的纪元, 分组交换网的出现成为现代电信时代的开始。

在电路交换中, 进行数据传输的两个结点之间必须先建立一条专用的通信路径(连接), 在通话的全部时间内用户始终占用端到端的固定传输带宽。

电路交换方式的优点是: 一旦连接建立后, 网络对用户是透明的, 数据以一固定速率传输, 传输可靠, 不会丢失、失序, 没有延迟。但是电路交换方式有时可能是非常浪费的。电路交换一般适用于系统间要求高质量的大量数据传输。

与电路交换不同的是存储转发交换方式, 采用这种方式, 使通信线路不为某对通信双方所独占, 在 ARPANET 中它是将整个要交换的信息报文(Message)分成若干信息分组(Packet), 对每个分组在通信子网中的每个交换结点上按存储—转发的方式在子网上传输, 因此把这种以存储—转发方式传输分组的通信子网又称为分组交换数据网(PSDN), 该方式大大提高昂贵的通信线路的利用效率。

根据传输的信息的大小, 存储转发方式分为分组交换和报文交换, 报文交换指将要传输的报文加上控制报头(如接收方的地址), 交给网络以存储转发方式传输。而分组交换是指将报文分成固定大小的分组, 加上控制头, 交由通信子网发送。与报文交换方式相比, 由于将报文分成更小的分组, 增加了分组在各个交换结点的并行传输, 分组传输总的延迟时间更短, 但由于每个分组要加上额外的头控制开销, 整个传输的开销增加。

分组交换又分为虚电路方式和数据报方式。所谓虚电路就是两个用户的终端设备在开始互相发送和接收数据之前需要通过通信网络建立逻辑上的连接, 一旦这种连接建立, 直至用户不需要发送和接收数据时清除这种连接。称这种方式为面向连接的。这种方法的优点是: 对于数据量较大的通信传输率高, 分组传输时延短, 且不容易产生数据分组丢失; 缺点是对网络依赖性大。数据报(Datagram)是将一个数据分组当作一份独立的报文看待, 每一个数据分组都含有源地址和目标地址信息, 交换结点须为每一个数据分组独立地寻找路径, 因此一份报文包含的不同分组可能沿着不同的路径到达终点, 而在网络终点需要重新排序。数据报的优点是对于短报文数据通信传输效率比较高, 对网络故障的适应能力强, 缺点是时延大。这种方式也称为无连接方式。

**【例 5-8】**网络层提供两种方式的服务, IP 协议提供的是(1)方式, ATM 提供的是(2)方式, 对于(1)方式, 分组在网上传输的开销与(2)相比要(3), 拥塞控制(1)比(2)要(4), (1)方式的健壮性是(5)。

**可选答案:**

- |         |         |         |        |             |
|---------|---------|---------|--------|-------------|
| (1)、(2) | A. 分组交换 | B. 电路交换 | C. 虚电路 | D. 数据报      |
| (3)     | A. 小    | B. 大    | C. 相同  | D. 不一定      |
| (4)     | A. 相同   | B. 困难   | C. 容易  | D. 根据不同情况决定 |

(5) A. 好 B. 不好 C. 网络线路决定 D. 路由器决定

正确答案:

(1) D (2) C (3) B (4) B (5) A

分析:

在网络层次体系结构中,网络层是通信子网的最高层,本题提到的交换方式是在网络层实现的,数据链路层(如 HDLC)提供的是点到点通信方式,网络层决定传输路由并根据交换方式控制信息的传输。所以,面向连接的服务和无连接的服务,或者称为虚电路方式和数据报方式,是网络层对上层提供的两种服务。

IP 协议是 TCP/IP 协议簇中属于网络层的一个协议,在双方进行数据交换前不需要建立连接,正因为如此,IP 协议可以连接不同的通信子网(面向连接的、无连接的)而很好地工作,IP 协议传输的信息也是划分为不同的 IP 数据报,因此,确切地说,IP 协议提供是数据报交换方式。

而 ATM 技术,在数据交换前要先通过 ATM 交换机建立虚电路,ATM 中的一个信元传输是通过在信元头的 VPI 和 VCI 字段上标志虚电路号进行的。在 ATM 交换机中,有一个虚连接表,无论是手工建立的永久连接,还是动态建立的连接,在该表中都有相应的项表示,结构如表 5-1 所示。

表 5-1 ATM 虚连接表项

入 口			出 口		
物理端口编号	VPI	VCI	物理端口编号	VPI	VCI
1	10	5	3	20	4
1	10	2	4	5	3

ATM 交换的信息是固定大小的 53 字节,其中 48 个有效负载和 5 字节的头,5 字节的头格式如图 5-4 所示。

字节

1	GFC		VPI	
2	VPI		VCI	
3	VCI			
4	VCI		PT	CLP
5	HEC			

(a) 用户网络接口

VPI		
VPI	VCI	
VCI		
VCI	PT	CLP
HEC		

(b) 网络结点接口

图 5-4 ATM 信元头部的结构

在信元传输过程中,用户网络接口 UNI (User Network Interface) (计算机—交换机)



和网络结点接口 NNI (Network Node Interface) (交换机—交换机) 两种信元的头部结构稍有不同。由此可见, ATM 传输方式是虚电路交换。

对于数据报方式, 每个分组都必须携带源地址和目的地址, 而虚电路方式, 数据分组 (或信元) 只要标志一个虚电路号即可, 所以传输同样多的数据, 数据报方式的网上实际信息传输量要比虚电路方式开销大。对于一条虚电路, 交换机可以控制通过这条虚电路的分组数目, 而数据报方式, 发送到同一目的地址的分组所走的路径可能不一样, 因此就无法控制, 所以数据报方式的拥塞控制要比虚电路方式困难。对于数据报方式, 因为分组是独立选路的, 当一条路径断开时, 分组可以从另外的路径到达对方, 而虚电路方式, 分组按同一条虚电路走, 当虚电路因故障而断开时, 需要重新建立虚电路才可以传输分组, 因此数据报方式的健壮性要强。

**【例 5-9】** 有多个设备可以实现不同网络或网段的互连, 工作在开放系统互连参考模型物理层、数据链路和网络层的互连设备分别称为 (1)、(2) 和 (3), Internet 上的防火墙可以工作在 (4) 层或 (5) 层。

可选答案:

- (1): A. 网关            B. 路由器            C. 协议转换器    D. 中继器  
(2): A. 转发器        B. 协议转换器    C. 网桥            D. 网关  
(3): A. 转发器        B. 路由器        C. 网桥            D. 中继器  
(4)、(5): A. 物理层   B. 数据链路      C. 网络层          D. 应用层

正确答案:

- (1) D (2) C (3) B (4) C (5) D

分析:

网络互连设备可以分为物理层互连、数据链路层互连、网络层互连和高层互连, 其中物理层互连设备是重发器, 也称中继器。数据链路层互连是网桥和交换机, 网络层互连是路由器, 高层互连通称网关。

在物理层上的互连设备其作用是增加网络的覆盖范围, 延长网线的距离, 互连设备做简单的信号接收、放大和发送, 不运行任何高层协议, 称之为中继器或重发器。重发器只能连接同一种类型的网络, 是为了增加传输线路的长度, 负责信号的再生和放大, 不能过滤数据。

数据链路层上的互连是连接两个网段, 起到两个网段的桥接作用, 称之为网桥。网桥实现两个网段的桥接, 同时也起到重发器的作用, 即延长线路距离和信号再生和转发。网桥连接的两个网段可以运行的 MAC 层协议是一致的, 也可以连接 2 个不同的 MAC 层网络, 如连接运行 802.3 协议的以太网和运行 802.5 协议的 Token Ring, 这时, 网桥要将从一个网段接收的 MAC 帧转换成另一个网段发送的另一个 MAC 帧, 这种情况下, 网桥实现的是协议转换器功能。

交换机原理上也工作在数据链路层, 可实现多个网段的信息交换, 网桥和交换机可

以隔离两个网段，防止在某一个网段的数据广播到另一网段，这是因为网桥接到数据帧后，判断接收到的数据帧目的地址是否和源地址在同一网段，如果在同一网段，就不再转发，而不是像重发器那样无条件转发，但网桥不能隔离广播数据报。

交换机被称为多端口网桥，交换机的每个端口可以连接1个网段。交换机有两种常用的交换技术，即直通式（Cut-Through）和存储转发式（Stored-Forwarding）。采用存储转发式时，集线器就像个分组结点交换机。它从一个输入端口收下一个帧，暂存后即根据其目的地址转发到适当的输出端口。直通交换不必将整个数据帧先缓存后再进行处理，而是在接收数据帧的同时就立即按数据帧的目的地址决定该帧的转发端口，这就使得转发速度大大提高。直通交换的一个缺点是它不进行差错检查就直接将帧转发出去，因此有可能也将一些无效帧转发给其他的站。有的交换机实现了两种方式的自动切换，以适应不同的通信线路的通信质量。

在数据链路层，网桥和交换机原理上只识别数据链路的帧格式，而路由器要识别网络层的分组格式，路由器工作在网络层，完成路由的选择等功能。现在出现的所谓三层交换机，是在交换机的基础上，增加了路由功能。

工作在第四层及以上层的互连设备通称为网关。网关根据不同的应用完成不同的功能，如服务代理、地址转换以及防火墙功能等。防火墙分为两大类：过滤型防火墙和应用级防火墙。过滤型防火墙可以由路由器完成，在网络层上进行IP地址过滤，也称IP级防火墙。IP级防火墙根据IP数据报的源IP地址、目的IP地址、源端口、目的端口及报文传递方向等报头信息来判断是否允许报文通过，即包过滤，在一台路由器上就可实现。包过滤一个很致命的弱点是不能在用户级别上进行过滤，即不能识别不同的用户和防止IP地址的盗用。如果攻击者把自己主机的IP地址设成一个合法主机的IP地址，就可以很轻易地通过包过滤器。

应用级防火墙在应用层根据不同的应用进行访问控制，需要面向不同的应用编制应用程序。它可以识别用户身份，限制某些应用通过，进行内外的中转使内外网络不直接相连等。

**【例 5-10】** 国际电信联盟的电信标准化部 ITU-T 的前身是(1)，其发布的 X.200 建议是和(2)制定的开放系统互连七层参考模型（OSI）等价的。作为最简单的防火墙—分组过滤器在该模型的(3)层检查出入地址；路由器是在该模型(4)层进行网络间中继的互连设备；FTP 则是 Internet 中常用的(5)层协议之一。

可选答案

- |          |         |        |          |        |
|----------|---------|--------|----------|--------|
| (1)、(2): | A. ANSI | B. ISO | C. CCITT | D. IEC |
| (3):     | A. 物理   | B. 网络  | C. 会话    | D. 应用  |
| (4):     | A. 物理   | B. 运输  | C. 数据链路  | D. 网络  |
| (5):     | A. 运输   | B. 会话  | C. 表示    | D. 应用  |

正确答案:

(1) C (2) B (3) B (4) D (5) D

分析:

本题要熟悉几个在网络和通信领域内的重要标准化组织, CCITT (国际电报电话咨询委员会) 就是一个重要的组织, 它制定了 X 系列和 V 系列的标准, 目前它被合并到国际电信联盟 ITU 中, 成为它的一个下属机构 ITU-T。ISO 是国际标准化组织, 它在信息处理方面制定了开放系统互连 OSI (Open System Interconnection) 七层参考模型。ANSI (American National Standards Institute) 是美国国家标准化协会。IEC (International Electrotechnical Commission) 是国际电工委员会。其他网络上常见的组织还有 IEEE (Institute of Electrical and Electronics engineers) (美国) 电气电子工程师协会, 它制定了局域网标准 802 协议。EIA (Electronic Industries Association) (美国) 电子工业协会, 它制定了标准中有著名的 RS-232C。另一个重要的是有关 Internet 的标准化组织 IAB (Internet Architecture Board) 因特网体系结构局, 著名的 IETF (Internet Engineering Task Force) 因特网工程特别任务组是其下设的一个特别任务组, Internet 的 RFC (Request For Comments) 大都出自工作组。

读者可能不熟悉 ITU-T 的 X.200 标准, 但并不影响本题的选择, 只要知道 OSI 参考模型是 ISO 的标准即可。

最简单的防火墙是包 (分组) 过滤防火墙, 它是根据 IP 地址进行过滤, IP 地址是第三层网络层用到的, 见上题所述。

路由器也是工作在网络层上的中继互连设备, 它用于连接两个不同的网络, 根据 IP 地址来决定转发的目的网络。路由器上运行 IP 协议进行 IP 数据包的转发工作, IP 协议对 IP 包的转发过程分为直接转发和间接转发, 间接转发是指将 IP 包发送到目的计算机时要经过中间的转发结点, 即发送的计算机和目的计算机不在一个子网内, 需要中转, IP 协议是通过判断本机和目的机是否属于同一子网号得出的。当数据到达最终的计算机所属的网络后, IP 协议就不能再通过中间结点转发了, 而是要找到该网络上的这台目的计算机, 这时, 还需通过 IP 地址找到 MAC 地址, 从而找到最终的计算机。这种转发方式称为直接转发。

同时, 运行 IP 协议的路由器在转发 IP 包时, 要根据路由表决定如何转发, 路由表的建立是通过运行路由协议完成的, 常用的路由协议有 RIP 和 OSPF 协议, 它们分别属于距离向量协议和链路状态协议。

FTP 是文件传输协议, 属于 TCP/IP 协议簇中的应用层协议。

【例 5-11】CSMA/CD 的访问控制方式是 IEEE 的 (1) 标准中制定的, 其中的 CSMA 是指 (2), CD 是 (3), 当侦听到冲突时, (1) 标准采用的是 (4) 继续侦听, 发现冲突后采用的退避算法是 (5)。



可选答案:

- (1): A. 802.2      B. 802.3      C. 802.4      D. 802.5  
(2): A. 码分多址访问      B. 令牌环访问控制  
      C. 载波侦听多路访问      D. 码分多路复用  
(3): A. 冲突检测      B. 码分      C. 激光唱盘      D. 呼叫设备  
(4): A. 非坚持      B. 1 坚持      C. 0 坚持      D. P 坚持  
(5): A. 二进制指数      B. 随机      C. 线性      D. 定时

正确答案:

- (1) B    (2) C    (3) A    (4) B    (5) A

分析:

IEEE 802.3 标准中制定的是 CSMA/CD 方式, 即带冲突检测的载波侦听多路访问, CSMA 指载波侦听多路访问 (Carrier Sense Multiple Access), CSMA 是指当一个站点要发送数据时, 先对总线进行监听, 看总线是否“忙”, 如果不忙就发送, 若“忙”, 就推迟发送, 直到总线不忙。CD (Collision Detection) 冲突检测, 是指在一个站点发送数据到总线后, 可能会跟别的站点发生冲突, 因为有可能别的站点也监听到总线不忙而发送数据, 因此需要冲突检测, 若检测到冲突, 则各个冲突站点马上停止发送, 并发送一冲突信号让其他站点尽快知道。然后随机延迟一段时间后, 再侦听信道状态。

在 IEEE 802.3 标准中, 侦听采用的 1 坚持方式, 即当侦听到线路忙时, 一直坚持侦听。其他还有非坚持和 P 坚持, 非坚持是指放弃侦听, 随机延迟一段时间再侦听。P 坚持是指当侦听到忙时以概率 P 侦听, 以  $1-P$  不侦听。当发现信道不忙而发送信息时, 发送完信息后还可能发生冲突, 因为可能还有发送站也侦听到线路空闲, 当检测到冲突, 发送站按二进制退避算法向后推迟。即发生冲突后, 退避时间扩大到上次的 2 倍, 当发生  $n$  次冲突时, 退避时间变成  $2^n$ 。

【例 5-12】以太网与令牌环相比较, 在重负载时以太网的性能 (1), 因为 (2)。在轻负载时, 从实时性来看, 以太网的性能是 (3), 在优先级的控制上, 令牌环方式 (4), 因为 (5)。

可选答案:

- (1)、(3): A. 好      B. 差      C. 一样      D. 不定  
(2): A. 冲突增加      B. 冲突减少  
      C. 分时传送, 无冲突      D. 根据应用要求决定  
(4): A. 困难      B. 容易      C. 与以太网一样      D. 不能控制  
(5): A. 访问是随机的      B. 帧格式支持      C. LLC 是一样的      D. 帧格式不支持

正确答案:

- (1) B    (2) A    (3) A    (4) B    (5) B



**分析:**

以太网方式采用的发送方法是,当有数据要发送时,要检测到信道空闲才发送,而令牌环方式要在一个空令牌转到发送站点时才能发送数据,所以在重负载时,以太网的冲突必然增加,整个网络性能下降,而令牌环方式因为其工作原理是分时传送,当重负载时,减少了令牌空转的时间,信道利用率反而提高。在轻负载时,以太网的冲突减少,一个站点有数据时就可以发送,而且成功率较高,而令牌环当一个站点要发送数据时,也必须等到空令牌转到该站点才能发送数据,所以以太网的实时性比较好。在优先级的控制上,令牌环方式下,通过在令牌格式中设置优先级,使得获得高优先级的站点具有优先发送的权力。

**【例 5-13】** 向端用户提供尽可能宽带的网络接入技术已引起人们的广泛关注。(1)只能提供 128Kbps 的接入数据速率,利用电话双绞线和光缆同轴电缆向端用户提供更高信息传输带宽的接入技术分别是(2)或(3),而(3)要配合电缆调制解调器(Cable Modem)。第三代无线通信可提供高达 2Mbps 的接入数据速率,采用的技术是(4)。光纤到户,即(5),则是将来的一种发展方向。

**可选答案**

(1)、(2): A. B-ISDN    B. N-ISDN    C. CDMA    D. ADSL

(3)、(4): A. HFC    B. GSM    C. CDMA    D. HDSL

(5):    A. FDDI    B. FTTH    C. FTTC    D. FTTB

**正确答案:**

(1) B    (2) D    (3) A    (4) C    (5) B

**分析:**

将用户的计算机接入 Internet 中,接入速度是衡量接入方式的一个主要因素,最初人们选择接入方式主要通过电话拨号方式,通过调制解调器进行数模转换,普通调制解调器 MODEM 的速度最高 56Kbps。N-ISDN 即窄带综合业务数字网可以向用户提供 2B+D 接口和 30B+D 两种接口方式,B=64Kbps,2B 可以达到 128Kbps。B-ISDN 即宽带 ISDN 是以 ATM 技术和光纤传输为基础的综合业务,能够提供更高的传输速率。

通过电话线(铜缆)接入达到高速传输采用的是 xDSL,包括高速数字用户线技术(HDSL)、非对称数字用户线技术(ADSL)、甚高比特数字用户线技术(VDSL)等,其中,最成熟和最常用的是 ADSL,一般下行信号速率可以达到 6.144Mbit/s,上行信号速率可以达到 640 Kbit/s。

电缆调制解调器(Cable Modem)用来处理来自于有线电视(CATV)服务商的模拟信号单元。光纤用户环路(FITL)包括了 FTTH(光纤入户)、FTTO(光纤到办公室)、FTTC(光纤到路边)、FTTB(光纤到大楼)等技术,HFC(光纤/铜轴混合接入)是光缆接入方式中的一种,在用户方使用现在的同轴电缆加 Cable Modem,在主干信道上使用光缆传输。

无线通信接入包括了 802.11、GSM/GPRS (Global System for Mobile Communication/General Packet Radio Service) 和 CDMA (码分多址), 传统 GSM 无线接口数据速率被限制在 9.6Kb/s, 而使用 GPRS 技术, 最大的数据传送速率可达 144Kb/s, CDMA 被称为第三代通信标准, 可提供高达 2Mbps 的接入数据速率。

FDDI 是光纤分布式数据接口 (Fiber Distributed Data Interface), 是城域网组网的一种技术。

【例 5-14】ADSL 对应的中文术语是 (1), 它的两种 Internet 接入方式是 (2) 接入。

可选答案

- |                  |               |
|------------------|---------------|
| (1) A. 分析数字系统层   | B. 非对称数字线     |
| C. 非对称数字用户线      | D. 异步数字系统层    |
| (2) A. 固定接入和虚拟拨号 | B. 专线接入和 VLAN |
| C. 固定接入和 VLAN    | D. 专线接入和虚拟拨号  |

正确答案

- (1) C (2) D

分析:

ADSL (Asymmetric Digital Subscriber Line) 翻译成中文为非对称数字用户线。ADSL 可以实现一对双绞线上传送高速数据和模拟信号, 为用户提供具有不同上下行数据传输速率的功能。一般下行信号速率可以达到 6.144Mbit/s, 上行信号速率可以达到 640 Kbit/s。目前, ADSL 已经逐渐成为广泛应用的 Internet 接入方式, 用户利用其提供的高速下行速率下载软件, 用较低的上行速率传送用户的命令控制信息。它把线路按频段分成语音、上行和下行三个信道, 故语音和数据可共用 1 对线。ADSL 特别适合于像 VOD 业务和 Internet 和多媒体业务的应用。

它的接入类型有两种:

- (1) 专线入网方式: 用户拥有固定的静态 IP 地址, 24 小时在线;
- (2) 虚拟拨号入网方式: 并非真正的电话拨号, 而是用户输入账号、密码, 通过身份验证, 获得一个动态的 IP 地址, 可以掌握上网的主动性。

【例 5-15】通过电话线连接因特网, 可以使用链路层协议有 SLIP 和 (1), 这种情况下给主机 (2) 一个 IP 地址。如果通过 N-ISDN 联网, 用户可以使用的信道带宽是 2B+D, 数据速率最大可达到 (3)。如果通过局域网连接因特网, 接入方式可以采用 ADSL, 最高下行速率可以达到 (4)。CHINADDN 是中国电信提供的数字数据网, 它采用 (5) 的交换技术为用户提供不同速率的专线连接。

- |                   |              |             |            |
|-------------------|--------------|-------------|------------|
| (1): A. PPP       | B. HDLC      | C. Ethernet | D. POP     |
| (2): A. 静态分配      | B. 动态分配      | C. 自动产生     | D. 不分配     |
| (3): A. 56Kb/s    | B. 64Kb/s    | C. 128Kb/s  | D. 144Kb/s |
| (4): A. 1.544Mb/s | B. 2.048Mb/s | C. 8Mb/s    | D. 10Mb/s  |

(5): A. 时分多路 B. 空分多路 C. 码分多址 D. 频分多路

正确答案:

(1) A (2) B (3) D (4) C (5) A

分析:

通过电话线拨号上网时,用到的链路层协议有 2 个,SLIP 和 PPP。SLIP 是使用较早的串行链路协议,由于其存在着诸多的限制,如只支持 IP 协议、不支持动态 IP 地址分配,保密性不好等原因,人们现在广泛使用 PPP 协议。PPP 协议克服了 SLIP 协议的缺点,具有进行错误检测,支持多种协议,连接时允许商议分配 IP 地址,允许身份验证等功能。

通过 ISDN 上网时,如果提供 2B+D 的接口,B 是 64Kbps, D 是 16Kbps,则总共可以提供 144Kbps 的带宽。

通过 ADSL 上网,下行信号速率最大可以达到 8 Mbit/s,上行信号速率可以达到 640 Kbit/s。

DDN(数字数据网)主要是以专线形式提供给用户固定出租线路业务,同时具有帧中继、压缩语音、虚拟专网等增值业务。DDN 提供的接口速率分为三类:2.048Mbps,  $N \times 64\text{Kbps}$  ( $N=1\sim 31$ ) 和 2.4/4.8/9.6/19.2 Kbps,当用户要求的接口速率是  $N \times 64\text{K}$  时,以时分复用方式(TDM)将 2.048M 速率复用给多个用户。当用户要求的接口速率是 2.4/4.8/9.6/19.2 Kbps 采用子速率复用方式将 64K 的速率复用给多个用户。

**【例 5-16】** 在 TCP/IP 协议中,有多个协议属于网络层,其中主要的协议是(1),它负责分组的路由选择和数据装配,(2)负责控制报文的发送,(3)负责网络地址到物理地址的映射,(4)负责物理地址到网络地址的映射,(2)协议分组在发送时要装配到(5)中发送。

可选答案:

(1) ~ (5): A. ICMP B. IP C. ARP D. RARP

正确答案:

(1) B (2) A (3) C (4) D (5) B

分析:

TCP/IP 的网络层主要协议是 IP 协议,它把传输层送来的报文组装成 IP 数据报,并把 IP 数据报传递给网络接口层。IP 协议制定了统一的 IP 数据报格式,以消除各通信子网的差异,从而为信息发送方和接收方提供透明的传输通道。IP 协议主要完成 IP 数据报在各个子网上的转发、IP 封装、分段和重组等功能。IP 协议的转发过程是通过获取 IP 报头的目的 IP 中的网络号,根据路由表中指定的转发端点获取转发到该端口的 MAC 地址,交由下层网络转发。

IP 协议为获取转发的端点的 MAC 地址,需要 ARP 协议的配合,ARP 是地址解析协议,用于将 IP 地址映射为物理地址,实际执行过程中,每台通信的机器有一个 ARP

表,表中有 IP 地址到 MAC 地址的对应项,开始 ARP 表为空。假设主机 A 要发送 IP 包到 B, A 根据就这个 ARP 表来找给定的主机 B 的 IP 对应的 MAC 地址,如果计算机 B 的 IP 不在 A 的 ARP 表上,就在网络上广播一个信息要求 IP 地址为 B 的计算机回答,拥有该 IP 地址的计算机 B 直接回答 A,告知其 MAC 地址。A 在其 ARP 表中添加此对应项,同时 B 也在自己的 ARP 中加入 A 的 IP 和 MAC 的对应项。如果在 A 的 ARP 表中有 B 的 IP 地址对应的 MAC,就根据 MAC 地址交数据链路层发送。这个 ARP 表有一个生命期,超过这个时期就被清除,这就可以反映网络的变化。RARP 是逆向地址解析协议,用于知道自己的物理地址情况下找到 IP 地址。现在基本被 DHCP (动态主机配置协议) 代替。

ICMP (Internet Control Message Protocol) 是差错和控制报文协议,在 IP 协议发现错误时,利用 ICMP 传输错误报告和控制信息。而在传送 ICMP 分组时,它是装配在 IP 分组中传送的。这两个协议是相互依赖的:IP 在发送一个差错报文时要用到 ICMP,而 ICMP 利用 IP 来传递报文。也就是 ICMP 报文被置于 IP 数据报的数据区中。然后这一数据报像通常一样被转发。

【例 5-17】Internet 是全球最大的、开放的计算机互连网络。网中每一台主机都分配有惟一的 (1) 位 IP 地址,其格式由 4 个小于 (2) 的数字组成,各数字之间由点号隔开。为了解决 IP 地址浪费问题,采用子网方式,在这种方式中,必须通过 (3) 来表示网络部分和主机部分,在数字表示中,是用 (4) 来标志数字是否是子网部分。子网划分方式对 IP 地址的影响是 (5)。

可选答案

- (1): A. 24      B. 32      C. 48      D. 64  
(2): A. 64      B. 128      C. 255      D. 256  
(3): A. 路由器    B. 计算机名    C. 域名      D. 掩码  
(4): A. 0      B. 1      C. 255      D. 256

(5): A. 可分配给主机的 IP 数量比原来增多    B. 可分配给主机的 IP 数量比原来减少  
C. 不产生影响    D. 子网数减少

正确答案:

- (1) B (2) D (3) D (4) B (5) B

分析:

直接连入 Internet 的计算机必须有一个惟一的 IP 地址,目前使用的 IP 协议的版本是第 4 版,称为 IPv4,其地址长度是 32 位,表示形式是分成 8 位为一组,8 位二进制数换算成 10 进制数,所以最大数只能到 255 (二进制 11111111)。

IP 地址由网络号和主机号 2 部分组成,分为 A. B. C. D. E 类, E 类保留不用, D 类表示组播地址,真正供分配的只有 A. B. C 类地址。各类地址的区别以 IP 地址最左边开始的 0、10、110、1110 和 1111 为标识。



A 类 IP 地址的网络号是 8 位, 主机号是 24 位。一个 A 类网络可以有  $2^{24}$  台主机, 除去主机部分为全“0”的表示网络地址的一个和主机部分为全“1”的表示广播地址的这两个地址外, 共有  $2^{24}-2$  个 IP 地址供分配。同样 B 类网络中一个网络号有  $2^{16}-2$  个 IP 地址供分配。C 类网络中一个网络号有  $2^8-2$  (254) 个 IP 地址供分配。

当得到一个 A 类或 B 类网络号时, 通常一个网络中没有这么多的主机, 另一方面, 又需要许多网络号, 因为每个上网单位需要不同的网络号。就出现了 IP 地址浪费问题和网络号不够分配问题, 为解决网络地址数不够问题, 采用子网划分方式, 将主机号的某几位拿出当作网络号, 这样就将一个大的网络划分成几个较小的网络, 就出现了子网的概念, 因此就不能靠 IP 地址类别来区分 IP 的网络部分和主机部分。为了使得 IP 协议能够识别出 IP 地址中, 哪些是表示网络号的位数, 哪些是表示主机号的位数, 就用一个 32 位的掩码 mask 来对应一个 IP 地址, 掩码位中“1”表示网络号, “0”表示主机号, 因此对于一个 IP 地址是 166.111.68.209 的计算机, 如果其掩码是 255.255.0.0 则网络号是 166.111 主机是 68.209, 如果掩码是 255.255.255.0 则网络号是 166.111.68 主机号是 209。通过掩码方式来确定网络号, 打破了原来的 IP 地址分类方式, 因此称为“无类域间选路” CIDR。

子网划分后, 分配给主机的 IP 地址数量要减少, 因为子网数量增加, 每个子网中的 2 个地址即全“0”和全“1”不能分配给主机使用, 它们是作为该子网的网络号和广播地址使用的。例如, 对于一个 C 类网络 192.168.1.0, 若不划分子网, 可以分配的主机数量从 192.168.1.1 到 192.168.1.254 共 254 个主机号。若划分为 4 个子网, 即 192.168.1.0 至 192.168.1.3, 对应的子网掩码为 255.255.255.192, 则每个子网的 6 位主机号部分的全“0”和全“1”部分不能分配给主机, 如 192.168.1.0、192.168.1.63; 192.168.1.64、192.168.1.127 等就不能作为主机号分配。由于每个子网有 2 个不能被分配的主机号, 当子网划分越多, 不能被分配的主机号就越多。

**【例 5-18】** 在 Internet 中要访问对方的计算机必须知道对方的 (1) 或 (2), 其中 (1) 不易记忆, 但 (2) 需要在网上有 (3) 提供相应的服务才能找到对方。为了找到一个计算机的物理位置, 要根据 (1) 找到 (4), 才能将数据帧送到最终目的地, 这个转换是使用 (5) 协议进行的。

可选答案:

- (1)、(2): A. 域名 B. IP 地址 C. Socket 端口号 D. 物理地址  
(3): A. Web 服务器 B. 数据库服务器 C. 域名服务器 D. 邮件服务器  
(4): A. 网络层地址 B. 计算机名称 C. 网络号 D. 物理地址  
(5): A. TCP B. IP C. ARP D. ICMP

正确答案:

- (1) B (2) A (3) C (4) D (5) C

分析:

在 TCP/IP 协议中, 根据 IP 协议的工作原理得知, 一台计算机要访问 Internet 上的另一台计算机必须得到对方的 IP 地址, 这样 IP 协议才可以将发往对方的数据根据 IP 地址送到。IP 地址是用数字表示的, 对人来讲, 很难记住对方的数字表示的 IP 地址, 例如我们经常访问的网站, 记住网站的名字容易, 但数字表示的 IP 不易记住, 因此就出现以字母表示的域名。

域名的意思是指 Internet 上的计算机有一个名字, 但这个名字是属于一个域的, 这样通过域和域内的名字可以分层管理 Internet 上的所有计算机。域名按树结构组织, 顶级域是 edu (教育机构), com (商业), gov (政府), org (非盈利组织), net (主要为网络商), int (一些 Internet 组织), mil (军队) 和标志国家的域名 (如 fr、gr、cn) 等。一个域是这个域名系统的一个子树, edu.cn 域名包括了像 tsinghua.edu.cn 这样的子域, 子域中包括了像 dream.tsinghua.edu.cn 这样的主机名, tsinghua.edu.cn 的注册机构被允许分配像 cs.tsinghua.edu.cn 这样的子域名并被委派可以在它下面分配名字, 如 mail.cs.tsinghua.edu.cn。

域名方便了人们识别不同的计算机, 但需要将域名转化为 IP 地址才能使 IP 协议得知与之通信的计算机。为了将域名转换为 IP 地址, 需要有域名服务器。由域名服务器负责解析域名和 IP 地址的对应关系。不同的域和子域一般都有负责本域域名解析的域名服务器 (DNS 服务器)。这样用户可以使用名字为 mail.cs.tsinghua.edu.cn 的计算机, 这个名字包括了机构名字 (tsinghua.edu.cn) 和计算机名字 (mail.cs)。要发送一个信息给用户, 你可以分别发送 zhangsan@ mail.cs.tsinghua.edu.cn 和 lisi@ mail.cs.tsinghua.edu.cn。你不用知道 mail.cs.tsinghua.edu.cn 在哪里, DNS 服务会将名字转换成地址, 如将 mail.cs.tsinghua.edu.cn 转换成 166.111.68.3, IP 协议就能找到一条到该机器的路径。

假设你发一个分组到 mail.cs.tsinghua.edu.cn, 本地目录服务就发送一个请求“寻找 mail.cs.tsinghua.edu.cn 的 IP 地址”给 cn 区域名字服务器, 该服务器知道 edu.cn 的域名服务器的地址并将请求转发给 edu.cn 的域名服务器, edu.cn 的域名服务器又知道 tsinghua.edu.cn 服务器, 就将请求转发给 tsinghua.edu.cn 服务器, tsinghua.edu.cn 服务器又知道 cs.tsinghua.edu.cn 服务器, 再将请求转发给 cs.tsinghua.edu.cn 服务器, 最后 cs.tsinghua.edu.cn 知道 MAIL 是在自己的域内, 就返回机器名为 mail 的 IP 地址给 tsinghua.edu.cn 服务器, 再上转给 edu.cn 服务器, 一直到最后发送到你的计算机上。整个过程中, 各服务器都缓存中间的结果, 例如 edu.cn 服务器缓存 tsinghua.edu.cn 的地址, 如果一个请求的地址已缓存在服务器中, 就可以直接作出回答用不着转发请求, 但缓存机制有时间限制以防止信息过期。

IP 协议选定了传输的下一步传输站点后, 要通过下层的协议来传输, 需要知道下一站点的物理地址 (MAC 地址), 需要地址解析协议 ARP 来完成该功能。

**【例 5-19】** 在日常操作中, 我们经常将重要的数据在另外的一个大容量的机器上保

存一个副本以防不测,例如上传到一个 FTP 服务器上,这种方式的应用模式叫做 A,在 Windows 操作系统中,有网上邻居的功能,这种方式的应用实际上是 (2) 模式,而在 Windows NT 和 Windows 2000 中,每台机器找到一个主域服务器或自己是独立域服务器,这种方式称为 (3)。当前的数据库系统有数据库服务器来调用,对数据库的操作不是在用户一方进行的,这种方式称为 (4),如果用户用浏览器来访问数据库的内容,必须有一个 Web 服务器,这种方式称为 (5)。

可选答案:

(1) ~ (3): A. 工作组 B. 文件服务器 C. 域模型 D. 客户/服务器

(4)、(5): A. B/S B. C/S C. B to B D. B to C

正确答案:

(1) B (2) A (3) C (4) B (5) A

分析:

在网络计算的几种模式中,文件服务器模式与域模式主要从对用户和资源的管理角度考虑,数据计算发生在每个用户的工作站(计算机)上。文件服务器、域服务器和工作组中的共享资源是作为数据备份的,服务器负责共享资源的管理。

文件服务器模式强调的是网络中有一个中心服务器,这个服务器硬盘容量比较大,可以给各个用户共享,用户在服务器的硬盘上存放自己的文件,用户也可以访问其他被共享的文件,所以称这样的服务器为文件服务器,服务器负责文件的管理、权限的分配等。

在文件服务器模式中,还有一种情况即工作组模式中,这种模式强调组内的计算机平等和共享,即每个组员可以将自己的文件设置为共享,供组内其他用户访问。网上邻居就是典型的工作组模式。

在 Windows NT 和 Win 2000 中,连网的计算机或者加入到一个已存在的域中,也可以成一个独立的域。域是由许多网络服务器与工作站连接而成的计算机群组,加入域中的计算机(服务器、工作站)等设备都成为域中的资源。每个域有一个域服务器负责整个域内的用户身份验证等,一个用户通过域服务器验证后,就可以域用户的权限访问域内其他机器上的资源,而不必在每台机器上都建立一个允许访问的用户。在多个服务器组成的域中,有一个服务器作为主域服务器。也可以将服务器划分成不同的域,称为多主域模型。多个不同的域之间可以建立信任关系,被信任域上的用户就可以访问信任域上的资源。

客户机/服务器模式主要从一次数据计算的完成过程这个角度而言的,客户机方进行数据请求,请求传到服务器,服务器负责完成数据计算或数据库的操作,最终结果返回到客户机。这种模式与文件服务器模式相比,最大的好处是数据计算发生在服务方,网络上只传输请求和计算结果,既减少网上的数据传输量,又能充分发挥服务器的高性能服务作用。这种方式与 Web 服务的进一步结合,出现了 B/S 应用模式。



【例 5-20】 Internet Explorer 是目前流行的浏览器软件，它的主要功能之一是浏览(1)。在浏览器主窗口的地址栏中输入想要访问的(2)的(3)或(4)地址并确认后，浏览器就开始在因特网上查找(2)的主页，一旦找到就可进行浏览。主页是用(5)语言书写的文件。

可选答案

- (1): A. 文本文件    B. 图像文件    C. 多媒体文件    D. 超文本文件  
(2): A. 端点    B. 站点    C. 起点    D. 终点  
(3): A. 域名    B. 用户名    C. 文件名    D. 目录名  
(4): A. LAN    B. WAN    C. IP    D. TCP  
(5): A. VC    B. C++    C. HTML    D. HTTP

正确答案:

- (1) D    (2) B    (3) A    (4) C    (5) C

分析:

浏览器软件识别的网页也叫超文本文件，超文本文件包括了文本文件、图像文件和多媒体文件。要访问一个网站（站点），即调用一个网站的主页，可以输入对方的 IP 地址，但一般情况下是输入网站的名字，称为域名。主页是用超文本标志语言 HTML 或各种脚本语言 Scripts 书写的文档。

【例 5-21】 防火墙是一种常用的网络安全装置，它可以(1)。有多种实现防火墙的技术，如包过滤、代理服务器、双穴主机和屏蔽子网网关等，相对来说(2)功能较弱但实现也较简单。

因特网的电子邮件、文件传输和 Web 访问中分别采用了(3)、(4)和(5)等协议。

可选答案

- (1): A. 防止内部人员的攻击  
      B. 防止外部人员的攻击  
      C. 防止内部人员对外部的非法访问  
      D. 既防止外部人员的攻击，又防止内部人员对外部的非法访问  
(2): A. 包过滤    B. 代理服务器    C. 双穴主机    D. 屏蔽子网网关  
(3)、(4): A. PPP    B. SMTP    C. FTP    D. WAP  
(5): A. HTTP    B. TCP    C. SNMP    D. ICMP

正确答案:

- (1) D    (2) A    (3) B    (4) C    (5) A

分析:

网络的路由安全从体系结构上看，可以从下层到上层采用不同的安全技术，其中防火墙也可以在 2 个不同的层次上实现。从整个网络的体系结构上看，在物理层可采取某些技术手段使得搭线偷听或检测变得不可能或不容易；数据链路层的点对点通信采用通



信加密技术；网络层采用包过滤、IP 安全协议（IP Sec）等措施；传输层采用传输安全协议（SSL 安全套接层协议）等；应用层采用安全的应用协议，如安全的 HTTP 协议 S-HTTP、邮件安全软件包 PGP（Pretty Good Privacy）等。

防火墙技术可以隔离内部网和外部网，因此可以防止外部人员对内部网的攻击，也可以限制内部人员对外部的访问，但不能防止内部人员的攻击。包过滤、代理服务器、双穴主机和屏蔽子网网关等技术是防火墙实现的具体技术方案。从协议的层次上大体上可以分两种：IP 级（网络层）防火墙和应用级防火墙。

IP 级防火墙是在 IP 层实现的，因此，它可以只用路由器完成。IP 级防火墙根据 IP 数据报的源 IP 地址、目的 IP 地址、源端口、目的端口及报文传递方向等报头信息来判断是否允许报文通过，即包过滤，在一台路由器上就可实现。包过滤一个很致命的弱点是不能在用户级别上进行过滤，即不能识别不同的用户和防止 IP 地址的盗用。如果攻击者把自己主机的 IP 地址设成一个合法主机的 IP 地址，就可以很轻易地通过包过滤器。

应用级防火墙是在应用层实现防火墙。其方式多种多样，包括应用代理服务器（Application Gateway Proxy）、套接字服务器（Sockets Server）、网络地址转换器（NAT Network Address Translator）等。应用级防火墙需要面向不同的应用编制应用程序。它可以识别用户身份，限制某些应用通过，进行内外的中转使内外网络不直接相连等。

网络中有各种不同的协议，了解这些协议首先要知道它属于哪个层次的协议，再需要知道协议的具体应用。

电子邮件系统使用 SMTP（Simple Mail Transfer Protocol）协议，系统由两个子系统组成：用户代理（User Agent）和消息传输代理（Message Transfer Agent），用户代理允许人们读取和发送电子邮件，消息代理是在后台运行的程序，在系统间传送电子邮件。SMTP 运行的前提是接收邮件的目的主机一直在运行，否则就不能建立 TCP 连接，如果用桌面计算机直接通过 SMTP 协议接收邮件是不现实的，因为桌面计算机每天要关机，不可能建立 SMTP 会话。这样，可以用一台 SMTP 服务器总是在线接收邮件，这个 SMTP 服务器提供邮件下载功能，桌面计算机与 SMTP 服务器交互，用 Client-Server 协议来检索消息，这样的协议之一就是 POP 协议，现在用 POP version 3（POP3）。

文件传输使用 FTP 协议，我们常用的 FTP 软件使用工具就是使用 FTP 协议。FTP 在 Client/Server 模式下工作，一个 FTP 服务器可同时为多个 CLIENT 提供服务，它总是等待 Client 系统向它提供服务请求，工作过程如下：（1）打开熟知端口（21），等待 Client 发连接请求，客户端可以用任意一个分配的本地端口号与服务器的 21 端口联系，这个进程称为主进程。（2）客户请求到来时，服务器启动子属进程来处理客户端发来的请求。（3）主进程返回，继续等待接收客户端发来的请求，与子进程并行工作。

Web 使用 HTTP（Hyper Text Transfer Protocol 超文本传送协议）协议，HTTP 的思想很简单，一个客户就像发一个邮件一样发送一个请求给服务器，服务器就像回答邮件请求一样发送一个回答消息给客户端。HTTP 下面使用的是 TCP 协议，先建立一条 TCP

连接, 连接是通过 TCP 提供的 SOCKET 端口完成的, 然后浏览器向服务器发送 GET 命令, 得到服务器传来的超文本文档。

其他几个协议, PPP 是点对点拨号通信协议, 属于数据链路层协议, 在家庭拨号上网中用到。WAP 是无线接入通信协议, 是对小型显示界面、低功率、小内存、CPU 运算能力低的通信工具设计的连接 Internet 的协议。SNMP (Simple Network Management Protocol) 是网络管理中最常用的网管协议。ICMP 是差错和控制报文协议, 见例题 16。

【例 5-22】 公开密钥方法的主要优点之一是 (1)。RSA 算法的基础是 (2)。当 N 个用户采用公开密钥方法保密通信时, 系统中共有 (3) 个密钥, 若采用传统的加密方法, 则有 (4) 个密钥, 为防止不老实的用户否认他们曾通过计算机发送过的文件, 较简便的方法是利用公钥方法完成 (5)。

可选答案

- |                |             |          |          |
|----------------|-------------|----------|----------|
| (1): A. 所有密钥公开 | B. 加密解密计算方便 |          |          |
| C. 便于密钥的传送     | D. 易于用硬件实现  |          |          |
| (2): A. 素因子分解  | B. 代替和置换的混合 |          |          |
| C. 求高阶矩阵特征值    | D. K-L 变换   |          |          |
| (3)、(4): A. N  | B. 2N       | C. $N^2$ | D. $N/2$ |
| (5): A. 文件加密   | B. 文件复制     | C. 数字签名  | D. 文件存档  |

正确答案:

(1) C (2) A (3) B (4) D (5) C

分析:

在数据的加解密中, 需要加密的信息称为明文, 这个明文信息 P 由一个加密函数 E 变换成密文 C, 这个函数以一个密钥 Ke 作为参数, 所以可以用  $C=E(P, Ke)$  来表达这个加密过程。解密过程也类似, 用一个解密函数 D 和解密密钥 Kd 对密文进行变换成为明文, 即  $P=D(C, Kd)$ 。加密算法可以分为两类, 即对称加密算法和非对称加密 (公钥) 算法, 对称加密算法中加密和解密采用同一个密钥, 即  $Ke=Kd$ , 称为对称密钥密码体制, 典型的代表是 DES。非对称加密算法中加密密钥和解密密钥是不一样的, 解密密钥只有接收方自己知道, 加密密钥公开, 称为公钥, 典型的代表是 RSA。

两种算法相比, 公钥算法的好处是便于密钥的传送, 但从计算的复杂性看, 公钥密码方法要比对称密钥方法复杂, 对称密钥方法已可以用硬件实现。

一般加密/解密的函数 (算法) 是公开的, 一个算法被破解的难度除了依赖于算法本身以外, 还往往与密钥长度有关, 通常密钥越长, 破解难度越高。

数据加密标准 DES 中使用的算法中, 输入为 64 位的明文, 使用 56 位的密钥, 输出是 64 位的密文, 使用 16 轮的混合操作, 使得密文的每一位都依赖于明文的每一位和密钥的每一位。目前已可利用专用芯片来实现 DES 算法。

公开密钥产生的主要原因除了解决对称密钥算法中的密钥分配问题外, 另一个需求

是对数字签名的需求。

在对称密钥算法中, 不管算法如何强壮, 一旦解密密钥被入侵者获得, 加密就没有意义了。一方面要将密钥保护好, 另一方面要分发给对方, 这就产生了一个两难的矛盾。而在公开密钥算法中, 加密密钥  $K_e$  是公开的, 解密密钥  $K_d$  是保密的, 公开密钥算法的特点如下: (1) 用加密密钥  $K_e$  对明文  $P$  加密后, 再用解密密钥  $K_d$  解密, 即可恢复明文, 即  $D_{K_d}(E_{K_e}(P)) = P$ , 此外  $E_{K_e}(D_{K_d}(P)) = P$ 。(2) 加密密钥不能用来解密, 即  $D_{K_e}(E_{K_d}(P)) \neq P$ 。(3) 从  $K_e$  推导出  $K_d$  是困难的。

RSA 算法实现的基本原理是找到两个大的素数, 加解密密钥是在此基础上计算出来的。要领是: 选择两个大整数  $p$  和  $q$ , 一般要求大于  $10^{100}$ , 计算  $n = p \times q$  和  $z = (p-1) \times (q-1)$ , 选择一个与  $z$  互质的整数, 记为  $d$ , 计算满足下列条件的  $e$ , 即  $(e \times d) \bmod z = 1$ 。对  $P$  加密就是计算  $C = P^e \bmod n$ , 解密则为  $P = C^d \bmod n$ 。这样, 加密密钥为  $(e, n)$ , 解密密钥为  $(d, n)$ 。这种方法的安全性在于分解一个大数 ( $n$ ) 的质因数 ( $p$  和  $q$ ), 并且计算出  $z$  的难度, 从理论上, 这是十分困难的。所以, 对 RSA 算法破解的难点是难以找到 2 个大的素数。

公开密钥体制中因为对每个用户都有 2 个密钥, 即公钥和私钥, 所以  $N$  个用户有  $2N$  个密钥。但每个用户只要小心保存好自己的 1 个解密密钥 (私钥) 即可。而传统的密钥体制中, 每对用户用一个密钥进行加密和解密, 所以共有  $N/2$  个密钥。

数字签名技术可以防止双方的抵赖和伪造, 可以进行通信双方的身份验证与鉴别。数字签名可以用消息摘要方法 MD 来实现, 也可采用公钥体制的加解密方法。

若用公钥体制的加解密方法, 例如, 发送者 A 用自己的解密密钥  $K_d$  对报文  $P$  进行运算, 将结果  $D_{K_d}(P)$  传送给接收者 B。B 用已知的 A 的公开加密密钥  $K_e$  得出  $E_{K_e}(D_{K_d}(P)) = P$ 。因为除 A 外没有别人具有 A 的解密密钥  $K_d$ , 所以除 A 外没有别人能产生密文  $D_{K_d}(P)$ , 这样报文  $P$  就被签名了。

若 A 要抵赖曾发送报文给 B, B 可将  $P$  及  $D_{K_d}(P)$  给第三者验证, 第三者很容易用  $K_e$  去证实 A 确实发送消息  $P$  给 B。反之, 若 B 将  $P$  伪造成  $P'$ , 则 B 不能在第三者前出示  $D_{K_d}(P')$ , 这样就证明了 B 伪造了报文。

## 5.3 思考练习题及答案

### 思考练习题

从网络上信息的传播方式看, Internet 上的信息传播是 (1), 而以太网上的传播方式属于 (2)。从拓扑结构上看, Internet 网络属于 (3), 从使用范围上看, 以太网一般是 (4), x.25 属于 (5)。

可选答案:

- (1)、(2): A. 点对点 B. 广播 C. 面向连接 D. 无连接方式  
(3): A. 完全连接 B. 不规则连接 C. 总线 D. 星型  
(4)、(5): A. 互联网 B. 专用网 C. 公用网 D. 分组交换网

计算机网络可以划分为资源子网和通信子网,属于通信子网的设备包括(6)和(7),电信部分负责的一般是(8)的建设,但如果它也提供内容服务,则也有(9)的建设内容。用户浏览网上内容的计算机属于(10)

可选答案:

- (6): A. 路由器 B. 服务器 C. 用户端主机 D. 应用程序  
(7): A. 终端设备 B. 服务器 C. 用户端主机 D. 通信线路  
(8)~(10): A. 资源子网 B. 通信子网  
C. 接入服务 D. 管理服务

(11) \_\_\_\_是控制通信交换的规则。

- A. 媒体 B. 标准 C. 协议 D. 以上全是

(12) OSI模型的哪一层制定了两结点间通信的规则?

- A. 运输层 B. 会话层 C. 数据链路层 D. 表示层

(13) OSI模型的哪一层进行检错并重发错误数据?

- A. 运输层 B. 物理层 C. 网络层 D. 应用层

(14) 在\_\_\_\_层决定同步点的位置。

- A. 运输层 B. 会话层 C. 表示层 D. 应用层

(15) 在\_\_\_\_层发生从一种字符码到另一种的转换。

- A. 运输层 B. 会话层 C. 表示层 D. 应用层

(16) 为什么要设计 OSI 模型?

- A. 生产商不喜欢 TCP/IP 协议 B. 数据传输率成指数增长  
C. 需要标准使两个系统通信 D. 协议容易实现

(17) 存储转发交换为什么比电路交换更有效?

- A. 在存储转发交换技术中,两个通信结点首先建立一个信道,然后开始传输,从而保证了可靠的连接并消除了重发的需要  
B. 在存储转发技术中,包可利用结点间的最快路由并且它到达的时间与数据流中其他包的到达的时间互相独立  
C. 在存储转发技术中,数据发送到一个中间结点,并在传输前重组,以便一起传输到目标结点  
D. 在存储转发技术中,根据交换机中的实时机制对数据包进行同步

(18) 以串行同步方式传送数据块时,经常采用的差错校验方法是\_\_\_\_。

- A. 偶校验码 B. 奇校验 C. 海明码校验 D. CRC 校验



(19) ARP 的功能是\_\_\_\_\_。

- A. 获得 IP 地址, 然后将 IP 地址映射到一个域名
- B. 获得网络上数据包的路由信息
- C. 获得 MAC 地址, 将 MAC 地址映射到 IP 地址
- D. 获得 IP 地址, 然后将 IP 地址映射到一个 MAC 地址。

TCP 协议是一个 (20) 协议, 标志双方通信是靠 (21), 加上本机的 IP 地址在 OSI 模型中被称为 (22), 而 UDP 是一 (23) 协议, 当用户要求 (24) 时, 要用 TCP 协议。

(20)、(23): A. 虚电路    B. 数据包    C. 面向连接    D. 无连接

(21): A. IP 地址    B. 网络地址    C. 套节字    D. 进程

(22): A. TSAP    B. PDU    C. 会话    D. 协议

(24): A. 快速传输    B. 可靠传输    C. 广播传输    D. 组播传输

TCP/ IP 是一个协议簇, 它的体系结构分为四层: 应用层、网际层、网络接口层和 (25), 其中 ARP 协议属于 (26)。为了在源主机和目的主机之间传送数据, IP 协议需要确定源主机和目的主机是否在同一个网络中。如果不在同一网络时, 则必须通过 (27) 进行通信。

(25) A. 会话层    B. 传输层    C. 网络层    D. 表示层

(26) A. 应用层    B. 传输层    C. 网际层    D. 网络接口层

(27) A. 网关或路由器    B. 中继器    C. 集线器    D. 终端匹配器

(28) NT 中 NETBEUI 协议的正确描述是\_\_\_\_\_。

- A. 可以通过不同的网络    B. 不能跨网段
- C. 适合大规模网络    D. 与 TCP/IP 兼容

(29) 5 类双绞线目前所能支持的最大吞吐量是多少?

- A. 10Mbps    B. 100Mbps    C. 1Gbps    D. 10Gbps

(30) 光纤为什么比铜线更安全?

- A. 它不传输电流, 因而更难以被窃听    B. 它更难以被切割
- C. 它能抗拒高压    D. 它不同时提供多种传输

(31) 网卡和驱动程序的功能是\_\_\_\_\_。

- A. 提供物理接口连接网络    B. 实现网络协议的物理层和数据链路层功能
- C. 执行 IP 协议    D. 执行 LLC 协议

(32) 网桥是如何知道它是应该转发还是应该过滤掉数据包的?

- A. 网桥维护过滤数据库, 该数据库基于数据包的目标地址能够识别哪些数据包该转发, 哪些该滤掉
- B. 从它传送的每一个数据包中解析出源结点地址; 所有不属于该网桥广播域的源结点地址都被过滤掉
- C. 网桥在目标结点重新请求前一直保护该数据包, 随后, 网桥就转发数据包

- D. 网桥把输入数据所用的协议与先前数据所用的协议作比较后, 过滤掉那些不匹配的输入数据包
- (33) 与集线器相比, 下面哪一个是使用交换机的优点?
- A. 交换机能够提供网络管理信息
  - B. 交换机能够给某些结点分配专用信道, 这使得数据传输更安全
  - C. 交换机能够更有效地从一个网段向另一个网段传输数据
  - D. 交换机能够在数据冲突发生率较高时提醒网络管理员
- (34) 对于帧的哪一个字段, 运行在直通方式下的交换机从来不读取它?
- A. 起始帧定界符
  - B. 源地址
  - C. 目标地址
  - D. 帧校验序列
- (35) EIA-232 定义了 DTE-DCE 接口的\_\_\_\_特性。
- A. 机械
  - B. 电气
  - C. 功能
  - D. 以上全是
- (36) 传输媒体的性能可以用\_\_\_\_衡量。
- A. 吞吐量
  - B. 传播速度
  - C. 传播时间
  - D. 以上全部
- (37) 在 ADSL 方式中, 最大的频宽用于\_\_\_\_。
- A. POTS
  - B. 上行通信
  - C. 下行通信
  - D. 以上全是
- (38) 使用 ISDN 的 D 信道有什么目的?
- A. 为了传输呼叫会话信息
  - B. 为了能够进行对称传输
  - C. 为了进行错误检测信息
  - D. 为了能够采用 TDM 技术
- (39) 帧中继和 ATM 连接有什么共同之处?
- A. 都借助于 Internet
  - B. 都利用了现有的 PSTN 线路
  - C. 都借助于虚拟电路
  - D. 小公司都能负担得起
- (40) 在一个 Token Ring 站点上会发生什么?
- A. 检查目的地址
  - B. 重新生成帧
  - C. 将帧传到下一个站
  - D. 以上全是
- (41) CSMA/CD 和 IEEE802.3 标准的另一个术语是\_\_\_\_。
- A. 以太网
  - B. Token Ring
  - C. FDDI
  - D. Token Bus
- (42) 100Base-T 的意义是\_\_\_\_。
- A. 100 兆频带传输速率
  - B. 100 兆基带双绞线传输
  - C. 在 100 米的范围内宽带传输
  - D. 在 100 米的范围内用双绞线传输
- (43) ATM 的信元共有\_\_\_\_。
- A. 53 字节
  - B. 48 字节
  - C. 53 比特
  - D. 48 比特
- (44) 路由器运行\_\_\_\_层的功能。
- A. 物理层
  - B. 数据链路层
  - C. 网络层
  - D. 运输层
- (45) Internet 是属\_\_\_\_类型的拓扑结构。
- A. 端到端
  - B. 环状
  - C. 网状
  - D. 分层

(46) 当一台主机知道它的物理地址但不知道 IP 地址, 它可以通过\_\_\_\_得到。

- A. ICMP      B. IGMP      C. ARP      D. RARP

如果通过局域网连接 Internet, 需要设置 TCP/IP 协议的属性, 其中需要指定 3 个 IP 地址, 即本机地址, (47) 地址和 (48) 的地址。

(47) A. 默认网关      B. 交换机      C. TCP 服务器      D. 远程访问服务器

(48) A. Web 服务器      B. 文件服务器      C. 邮件服务器      D. DNS 服务器

(49) 目前仍在使用的最简单的一种最简单的网络形式是什么?

- A. 基于服务器的网络      B. 瘦客户机网络  
C. 主机到主机网络      D. 端到端网络

(50) B/S 结构中客户访问数据库与 C/S 结构中的客户访问数据库的区别在于\_\_\_\_。

- A. B/S 结构中 Web 服务器直接执行 SQL 语句, 而 C/S 结构中的服务器通过 CGI 方式。  
B. B/S 结构中 Web 服务器可以通过 CGI 方式, 而 C/S 结构中的服务器直接执行 SQL 语句。  
C. B/S 结构中客户直接执行 SQL 语句, 而 C/S 结构中的客户通过 CGI 方式。  
D. B/S 结构中客户可以通过 CGI 方式, 而 C/S 结构中的客户直接执行 SQL 语句。

(51) 下列关于加密的叙述中, 正确的是\_\_\_\_。

- A. DES 属于公钥密码体制  
B. RSA 属于公钥密码体制, 其安全性基于大数因子分解困难  
C. 公钥密码体制的密钥管理复杂  
D. 公钥密码体制中, 加密和解密采用不同的密钥, 解密密钥是向社会公开的

(52) RSA 加密方法的成功之处在于找到\_\_\_\_很难。

- A. 公钥      B. 公钥的素因子      C. N      D. N 的素因子

(53) 在公钥体系的加解密方法中, 哪个密钥是公开的?

- A. 只有加密密钥      B. 只有解密密钥  
C. 加密和解密密钥      D. 没有

(54) 下面哪一个最好地描述了代理服务器的功能?

- A. 拒绝对特殊 IP 地址的局域网访问  
B. 滤去准备从互联网传输到内部网的不适当内容  
C. 以 IP 格式压缩协议  
D. 充当内部网与外部世界间的网关对内部 IP 地址进行隐藏

(55) 利用公开密钥算法进行数字签名时, 发送方签名用的是\_\_\_\_。

- A. 接收方的公开密钥      B. 发送方的公开密钥  
C. 接收方的秘密密钥      D. 发送方的秘密密钥

(56) 对称密钥算法与公开密钥算法相比,其好处是\_\_\_\_\_。

- A. 密钥保密容易                      B. 算法运算速度快  
C. 算法简单                              D. 算法公开

(57) OSI (Open System Interconnection) 安全体系方案 X.800 将安全服务定义为通信开放系统协议层提供的服务,用来保证系统或数据传输有足够的安全性。X.800 定义了五类可选的安全服务。下列相关的选项中不属于这五类安全服务的是\_\_\_\_\_。

- A. 数据保密性    B. 访问控制    C. 认证    D. 数据压缩

(58) 数字签名技术可以用于对用户身份或信息的真实性进行验证与鉴定,但是下列的\_\_\_\_\_行为不能用数字签名技术解决。

- A. 抵赖              B. 伪造              C. 篡改              D. 窃听

当  $n$  ( $n \geq 1000$ ) 个用户采用对称密码进行保密通信时,任意两个用户之间都需要一个安全的信道,系统中共有 (59) 个密钥,每个用户需要持有 (60) 个密钥;而当  $n$  个用户采用公钥密码方法进行保密通信时,共有  $2n$  个密钥,每个用户需要持有 (61) 个密钥(公开的、可任意使用的公钥不算在内)。

(59) A.  $n$               B.  $2n$               C.  $n(n-1)/2$               D.  $n(n-1)$

(60) A.  $n-1$               B.  $n$               C.  $2(n-1)$               D.  $2n$

(61) A. 1              B. 2              C.  $n-1$               D.  $2n$

(62) 在网络排除故障时不会经常使用的命令是\_\_\_\_\_。

- A. ping    B. netstat    C. tracert    D. dir

(63) 一台服务器用 IP 地址可以 ping 通,但用域名却 ping 不通,原因可能是\_\_\_\_\_。

- A. 服务器未启动              B. 服务器未运行 TCP/IP 协议  
C. DNS 配置不对              D. 服务器负荷太重,无时间应答

### 思考练习题答案

- (1) A (2) B (3) B (4) B (5) C (6) A (7) D (8) B (9) A  
(10) A (11) C (12) C (13) A (14) B (15) C (16) C (17) B  
(18) D (19) C (20) C (21) C (22) A (23) D (24) B (25) B  
(26) C (27) A (28) B (29) B (30) A (31) B (32) A (33) C  
(34) D (35) D (36) D (37) C (38) A (39) C (40) D (41) A  
(42) B (43) A (44) C (45) C (46) D (47) A (48) D (49) A  
(50) B (51) B (52) D (53) A (54) D (55) D (56) B (57) D  
(58) D (59) C (60) A (61) A (62) D (63) C



## 第 6 章 程序设计语言基础

本章分两部分：程序设计语言的基础知识和程序语言处理程序的基础知识。

### 6.1 内容提要

#### 6.1.1 程序语言基础知识

程序语言的基础知识包括两方面的内容，一方面是程序语言的基本概念，包括有代表性的常用的程序语言的数据类型、控制结构，函数定义与函数调用，程序结构等；另一方面是对于有代表性的程序语言，像 FORTRAN, ALGOL, PASCAL, C, C++, Java, COBOL, LISP, PROLOG 等，它们的应用领域是什么，它们的数据类型、语句、程序结构等方面有什么特点。

数据类型是对数据的描述，数据的分类是根据数据的书写格式及在其上的操作进行的。因此，一个数据类型是由值集合以及在这个值集合上的操作集合构成的。

指针类型数据的值是存放数据对象的内存地址，在程序语言中是变量的地址，含义是指向这个数据对象。

可以把类(class)看做一种新的数据类型，它扩充了结构体(struct)和共用体(union)，使之具有函数成员。

类描述了一组相似对象的共同特性，这一组相似对象被称为该类的实例。类作为一种模式，对象是具有这种模式的具体例子。类的接口提供了用户可以使用的操作，类的体则定义了这些操作的具体实现，其细节对外不公开，从而保证了类的接口是一种抽象。

程序语言所提供的控制结构一般在三个级别上，它们是表达式、语句和函数。理论上证明了可计算问题的程序在语句级都可用顺序、选择和重复这三种控制结构来描述。

#### 6.1.2 语言处理程序概述

对高级语言程序的处理有两种方式，解释和编译，相应的程序分别称为解释程序（或解释器）和编译程序（或编译器）。

如果源程序是汇编程序，那需要一个汇编器，它把汇编程序翻译成等价的机器语言程序，如图 6-1 所示。

读者应掌握解释程序的构成和 workflows，组成编译程序的各个阶段以及用解释方式和编译方式处理高级语言程序的区别。

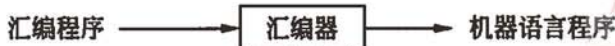


图 6-1 汇编器的功能

### 6.1.3 构造编译程序基本知识

编译程序对源程序的处理，首先是分析被加工对象是什么，然后将其转换成目标程序，这是程序处理信息的一般步骤。构造编译程序的基本原理和技术，也可用来构造解释程序和汇编器。

读者应掌握程序语言的形式定义以及从定义构造出识别器的方法。对于词法分析，要掌握用正规式定义一个程序语言的单词集合以及从正规式构造出确定的最小化的有限自动机。对于语法分析，掌握一个程序语言语法的形式定义，从这个定义出发，构造出自上而下或自下而上的分析器。

语义分析和转换是在语法分析的框架上扩充而成的，首先把每个语法符号加上语义信息，在语法分析的框架上挂上语义分析和转换的函数。随着语法分析的一步步展开，每识别出一个语法单位，就调用相应的语义分析和转换函数，完成相应的语义分析和翻译的任务，这就是所谓的语法制导翻译。

另外，还应掌握中间代码的形式，中间代码优化和生成目标代码的有关知识。

## 6.2 例题分析

### 6.2.1 程序语言基础知识

【例 6-1】 程序设计语言的定义一般包括 (1) 几个方面。程序设计语言可划分为高级语言和低级语言两大类。与高级语言相比，用低级语言开发的程序，其 (2)，用在 (3) 的场合，它使用 (4) 提高程序的可读性。

可选答案：

- |                    |                |
|--------------------|----------------|
| (1) A. 语法、语义和语句    | B. 语法、语义和语用    |
| C. 语义、语句和语用        | D. 语法、语用和语句    |
| (2) A. 运行效率低，开发效率低 | B. 运行效率低，开发效率高 |
| C. 运行效率高，开发效率低     | D. 运行效率高，开发效率高 |
| (3) A. 对时间和空间有严格要求 | B. 并行处理        |
| C. 事件驱动            | D. 电子商务        |
| (4) A. 简单算术表达式     | B. 助记忆符号       |
| C. 伪指令             | D. 定义存储语句      |

**分析:**

程序设计语言用以书写计算机程序(指计算任务的处理对象和处理规则的描述),它包括语法、语义、语用三个方面。语法表示程序的结构或形式,即表示构成语言的各种记号间的组合规则,像(对于高级语言来说)词、表达式、句子、函数乃至程序是如何构成的。但不涉及这些记号的特定含义,也不涉及使用者。语义表示程序的含义,即表示按照各种方法所表示的各个记号的特定含义,但不涉及使用者。语用表示程序与使用者的关系。

程序设计语言的基本成分有数据、运算、控制和传输。数据成分用以描述程序中所涉及到的数据;运算成分用以描述程序中所包含的运算;控制成分用以表达程序中运算的执行顺序;传输成分用以表达程序中数据的传输。

可以从不同的角度对程序设计语言进行分类,从程序语言的抽象层次上来看,可以分为三类:机器语言、汇编语言和高级语言。

机器语言是特定计算机系统所固有的语言,用机器语言编写的程序可读性很差,程序员难以修改和维护。

汇编语言用助记符号来表示机器指令中操作码和操作数,汇编语言仍然是一种和计算机的机器语言十分接近的语言,它的书写格式在很大程度上取决于特定计算机的机器指令。

用低级语言(机器语言或汇编语言)进行程序设计,可以充分发挥人的聪明才智,针对所解决的问题,在最大程度上利用计算机的资源,例如,巧妙安排存储,合理使用 cache、寄存器、效率高的指令等。但是,开发维护软件的费用太高。

随着计算机科学的发展,人们开发了功能强、抽象级别高的程序设计语言以支持各个领域、各种程序设计方法,于是就产生了面向各类应用和支持不同程序设计方法的程序语言,称为高级语言。目前已开发出许多高级语言,常见的有 FORTRAN、COBOL、ALGOL、PASCAL、C、ADA、C++、Java 等。这类语言与人们使用的自然语言比较接近,大大提高了程序设计的效率。

**正确答案:**

(1) B (2) C (3) A (4) B

**【例 6-2】**一种最早用于科学计算的程序设计语言是(1);一种提供指针和指针操作且不存在布尔类型的、应用广泛的系统程序设计语言是(2);最早用 BNF 描述其语法的程序语言是(3);一种适合在互联网上编写程序可供不同平台上运行的面向对象程序设计语言是(4);一种在解决人工智能问题上使用最多的、有强的表处理功能的函数程序设计语言是(5);一种以谓词逻辑为基础,核心是事实、规则和推理机制的逻辑程序设计语言是(6)。

**可选答案:**

(1) ~ (6): A. PASCAL      B. ADA      C. SMALLTALK      D. SNOBOL

E. C                      F. ALGOL   G. Java  
I. PROLOG                J. FORTRAN

H. LISP

分析:

FORTRAN 是第一个被研制出来的高级程序设计语言,以后发展起来的若干 FORTRAN 版本,广泛用于书写科学计算程序。

C 语言的主要特色是兼顾了高级语言和汇编语言的特点,简洁、丰富、可移植。C 与 UNIX 操作系统紧密相关,UNIX 操作系统及其上的许多软件都是用 C 编写的。C 有丰富的类型定义系统,指针的使用更灵活,例如可把数组名看成一个指针常量;一个指针可以指向一个数组的开始,一行或一个元素。

ALGOL 60 主导了 20 世纪 60 年代程序语言的发展。它有严格的文法规则,第一个采用巴科斯范式 (BNF) 来描述。

Java 产生于 20 世纪 90 年代,其目的是用于开发网络浏览器的小应用程序,但是作为一种通用的程序设计语言,Java 也得到了广泛的应用。Java 保留了 C++ 的基本语法、类和继承等概念,删掉了 C++ 中一些不好的特征。

LISP 是一个函数式语言,它以  $\lambda$ -演算为基础,多应用于人工智能领域。函数是一种对应规则 (映射),它使定义域中每个元素和值域中惟一的元素相对应。

PROLOG 是一个逻辑型语言,建立在关系理论和一阶谓词理论上。它的程序是一系列事实、数据对象或事实间的具体关系和规则的集合,关键操作是模式匹配,通过匹配一组变量与一个预先定义的模式并将该组变量赋给该模式来完成操作。PROLOG 有很强的推理功能,适用于书写自动定理证明、专家系统、自然语言理解等问题的程序。

正确答案:

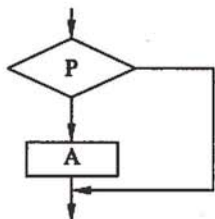
(1) J   (2) E   (3) F   (4) G   (5) H   (6) I

【例 6-3】一般程序语言中提供了描述 (1), (2), 控制和数据传输的语言成分。控制成分中有顺序结构、选择结构和重复结构,重复结构可用 (3) 和 (4) 中的流程图表示。一般程序语言中也提供了 GOTO 语句,但是,结构化程序设计要求 (5) 它。

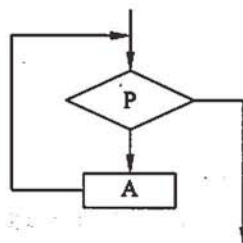
可选答案:

(1)、(2): A. 数据                      B. 整型                      C. 表达式                      D. 计算  
                    E. 过程                      F. 数组

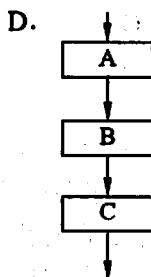
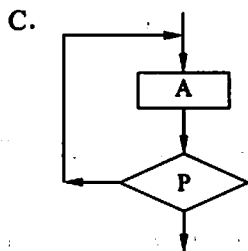
(3)、(4): A.



B.







(5): A. 绝对不使用 B. 尽量少使用 C. 随便使用 D. 尽量多使用

分析:

一个程序的最基本部分是对数据和对数据操作(计算)的描述。

循环结构描述了控制重复计算的规则,通常由三部分组成:初始化、重复部分 A 和重复条件 P。重复结构主要有两种形式: while 型结构和 do-while 型结构。while 型结构是先判断条件 P,若成立,则执行 A,然后再去判断重复条件 P;否则控制流程就退出重复结构,如 B 所示。do-while 型结构的是先执行 A,然后再判断条件 P,若成立则继续执行 A;否则控制流程就退出重复结构,如 C 所示。请注意,do-while 型结构中 A 至少被执行一次;而 while 型结构 A 可能一次都不执行。

从对算法的逻辑控制来说,不用 goto 语句也是可以完成的,在有些情况下,使用 goto 语句能提高执行效率,但程序中出现 goto 语句,会使程序的书写顺序和程序的执行顺序不一致,降低程序的可读性。因此,应尽量少使用它。

正确答案:

(1) A (2) D (3) B (4) C (5) B

注:(3)和(4)的选择可交换。

【例 6-4】在 C 语言中,若函数调用时实参是数组名,则传递给对应形参的是(1)。在下面的程序中,若实参 a 与形参 x 以值调用(call by value)的方式传递信息,那么输出结果为(2);若实参 a 与形参 x 以引用调用(call by reference)的方式传递信息,则输出结果为(3)。

```

main ( )                                void addone (int x)
{   int a;                               {   int a;
    a=1;                                  a=x+1;
    addone (a);                           x=x+2;
    printf ("%d",a);                      return;
}                                           }
  
```

可选答案:

(1): A. 数组空间的首地址 B. 数组的第一个元素值  
C. 数组中元素的个数 D. 数组中所有的元素

(2)、(3): A. 1 B. 2 C. 3 D. 4

分析:

在C语言中,数组名代表该数组的起始地址,即是说,数组名是一个指针,它的值是该数组的起始地址,它指向该数组的开始位置。不过,它不能被修改。在函数的定义中,可以用数组名作函数参数,被声明为数组的形参实际上是一个指针。当实参向形参传递数组时,按值调用传递数组的首地址,数组元素本身不被复制。因此,如果函数调用时实参是数组名,则形参指针接受到的是实参数组的首地址。

传值调用是指把实在参数的值传递给相应的形式参数,子程序不能通过这种方式传回任何结果。

引用调用是指把实在参数的地址传递给相应的形式参数,此时子程序对形式参数的一次引用或赋值都是对形式参数的间接访问。

实参a与形参x以传值调用的方式传递信息,相当于执行 $x=a$ 。控制进入函数addone后,形参x的值是1,执行“ $a:=x+1$ ;”和“ $x:=x+2$ ;”后,函数addone中a的值更新为2,x的值更新为3,而主函数中的a的值保持不变。可以看出,若实参a与形参x以传值调用的方式传递信息后,在控制进入被调用函数后,执行被调用函数体访问的是形参x,与main中的实参a无关。

实参a与形参x以引用调用的方式传递信息,相当于执行 $x=&a$ 。控制进入addone后,执行语句“ $a=x+1$ ;”和“ $x=x+2$ ;”,相当于执行“ $a'*=x+1$ ;”和“ $*x*=x+2$ ;”, $*x$ 即是a,也可表示成“ $a':=a+1$ ;”和“ $a=a+2$ ;”。请注意,a'是函数addone中a,a是主函数中的a。因此,主函数中输出的值为3。

正确答案:

(1) A (2) A (3) C

【例 6-5】在面向对象的方法中,对象可看成是属性(数据)以及这些属性上的专用操作的封装体,封装是一种(1)技术,封装的目的是使对象的(2)分离。

类是一组具有相同属性和相同操作的对象的抽象描述,类的每个对象都是这个类的一个(3)。类之间共享属性和操作的机制称为(4)。一个对象通过发送(5)来请求另一对象为其服务。

可选答案:

- |               |          |          |          |
|---------------|----------|----------|----------|
| (1): A. 组装    | B. 产品化   | C. 固化    | D. 信息隐蔽  |
| (2): A. 定义和实现 | B. 设计和测试 | C. 设计和实现 | D. 分析和定义 |
| (3): A. 例证    | B. 用例    | C. 实例    | D. 例外    |
| (4): A. 多态性   | B. 动态绑定  | C. 静态绑定  | D. 继承    |
| (5): A. 调用语句  | B. 消息    | C. 命令    | D. 口令    |

分析:

对象:在现实世界中,对象无处不在,人们通过对象来思考问题。所有对象都有属

性 (attributes) 和行为 (behaviors), 例如电视机, 有音量、亮度、辉度、频道等属性, 可以有切换频道、调整颜色、增大或降低音量等操作。人们通过观察对象的属性和行为而研究对象。面向对象程序设计用软件对象模拟实际对象, 将数据 (属性) 和函数 (或行为或操作或方法) 封装成对象。

我们所说的对象是软件对象 (又称计算机对象), 每个对象有它自己的属性值, 表示该对象的状态。对象中属性值只能通过该对象提供的操作来访问。操作也称做方法或服务, 它规定了对象的行为, 表示对象所提供的服务。对象具有信息隐蔽性, 用户只能看见对象接口界面上的信息, 不能访问对象的“私有数据”, 也看不见实现操作的具体细节。封装是一信息隐蔽技术, 它的目的是使对象的使用者和对象的生产者分离, 使对象的定义和对象的实现分离。

对象并不是孤立存在的, 彼此之间通过传递消息进行交互。这样, 对象可表示成三元组 (接口、状态、操作)。

类是由用户定义的数据类型, 它将具有相同状态操作和访问机制的多个对象抽象成一个对象类。在定义了类以后, 属于这种类的一个对象叫作类实例或类对象。一个类的定义应包括类名、类的成员和访问规则、类的实现。一般形式如下:

```
class    类名
{
    类成员 1;
    类成员 2;
    ...;
    类成员 n;
}
```

其中, 类成员或者是一个数据声明, 或者是一个方法声明。数据对象声明可以是 const 声明、struct 声明、任何类型或类的数据对象声明。方法声明是函数或过程的定义。成员表可以省略, 这样的类叫空类。

类的每个成员都具有描述该成员可见性的访问控制属性, 它可能是 private 属性、protected 属性或 public 属性。

**private (私有的):** 说明类的成员是私有的, 只能被该类的成员函数和友元函数 (用 friend 关键字表明, 它不是类的成员函数) 访问。

**public (公有的):** 说明类的成员是公有的, 它不仅可以被该类的成员函数访问, 而且还可以被类的外部访问。

**protected (被保护的):** 说明类的成员是被保护的, 它只能被该类的成员函数、友元函数和该类的派生类的成员函数访问。

类和对象 (类实例) 的关系就像类型和值的关系一样。类对象必须先说明后使用。类对象的说明形式如下:

类名 对象名;

类对象的成员可用直接成员运算符“.”和间接成员运算符“→”表示成:

对象名.成员名 或 对象名→成员名

构造函数和析构函数都是类的成员函数,它们可由系统隐含说明,也可由用户定义。构造函数与类同名,在定义类对象时,自动调用它,为对象分配内存,进行必要的初始化和类型转换工作。析构函数在类名前加一个波浪号“~”,它不带任何参数,当局部对象超出作用域时,析构函数被隐含地调用,释放对象占用的内存。例如,

```
class stack{
    private:
        int buffer[stacksize];
        int * sp;
    public:
        stack ( );          /*构造函数*/
        ~stack ( );         /*析构函数*/
        void push (int x ); /*成员函数*/
        void pop ( );
        int top ( );
};
```

继承是类之间的基本关系,它是基于层次关系的不同类共享数据和操作的一种机制。程序员在建立一个新类时,可以让新类继承已定义基类的数据成员和成员函数。这个新类称为派生类,派生类通常添加了其自身的数据成员和成员函数,因而通常比基类大得多。派生类比基类更具体,它代表一组外延较小的对象。

继承使类形成树状层次结构,基类和它的派生类构成了一种层次关系。所以,基类也称父类,派生类也称子类。

继承是软件复用的一种形式。

消息:消息传递是对象间通信的手段。一个对象通过向另一个对象发送消息来请求其服务。一个消息通常包括接收对象名、调用的操作名以及适当的参数。消息告诉接收对象完成什么操作,接收对象分析接收消息,完成相应的操作。

多态性:多态性是指通过继承相关的不同类,它们的对象能够对同一个函数调用做出不同的响应。例如,从基类四边形派生出矩形类和正方形类,矩形和正方形是四边形,对四边形的操作(例如,计算周长和面积)也能用在矩形和正方形上。因此,我们可以针对四边形定义一个计算周长的函数f,当函数f作用在一个矩形a上时,矩形a将计算自己的周长;当函数f作用在一个正方形b上时,正方形b将计算自己的周长。在C++中,我们在基类四边形中把f定义成虚函数,在矩形类和正方形类中重新定义函数f。



利用 C++ 的多态性和虚函数机制，能设计并实现易于扩展的软件系统。

正确答案：

(1) D (2) A (3) C (4) D (5) B

## 6.2.2 语言处理程序概述

【例 6-6】判断下面的陈述是否正确。

- (1) 解释程序是接受参数，按照某一样板产生机器语言程序的计算机程序。
- (2) 编译程序是为把高级语言书写的程序翻译成面向计算机的目标程序而使用的计算机程序。
- (3) 就处理速度而言，编译方式比解释方式慢。
- (4) 解释程序是按照源程序的控制流程逐句分析执行源程序语句的计算机程序。
- (5) C 和 FORTRAN 语言程序通常解释执行。Java 类似 C++，和 C++ 一样都采用编译方式。
- (6) 使用编译程序时，因为是逐句地翻译执行源程序的语句，所以可逐条语句排错。
- (7) 汇编程序的功能是将汇编语言书写的源程序翻译成由机器指令和其他信息组成的目标程序。
- (8) 任何一种汇编语言的每一语句都能对应于一条机器语言指令。

分析：

任何一种高级语言（如 C 语言）都精确规定了数据结构和程序的执行顺序，即定义了一台计算机（称做 A）。人们完全能用硬件设计制造计算机 A，它的存储结构是 C 语言的数据结构，它的控制器控制 C 语言程序的执行，它的运算器完成 C 语言的语句操作，A 的机器语言即是 C 语言，每个 C 语言程序都规定了计算机 A 从初始状态到终止状态的转换规则。

不过，设计制造这样一台计算机的费用太大了，我们用一台通用计算机 B 来模拟计算机 A 的执行，为达到这个目的，必须用计算机 B 的机器语言构造一组程序以支持用 A 的机器语言 C 编写的程序的执行。换句话说，我们在一台通用计算机 B 上用软件构造了一台高级语言计算机 A。这个过程称为软件模拟（或软件解释）。我们把模拟计算机 A 的软件称为虚拟计算机（或抽象计算机）。

解释程序的工作过程如图 6-2 所示，它按输入程序的控制流程解释执行，产生程序规定的输出。

编译程序的功能是把某高级语言书写的源程序翻译成与之等价的低级语言（汇编语言或机器语言）程序，从编译的角度称它为目标程序。采用编译方式处理高级语言程序，要分成两个阶段，第一个阶段叫编译，把源程序翻译成与之等价的的目标程序；第二个阶段叫运行，执行目标程序得到输出结果。

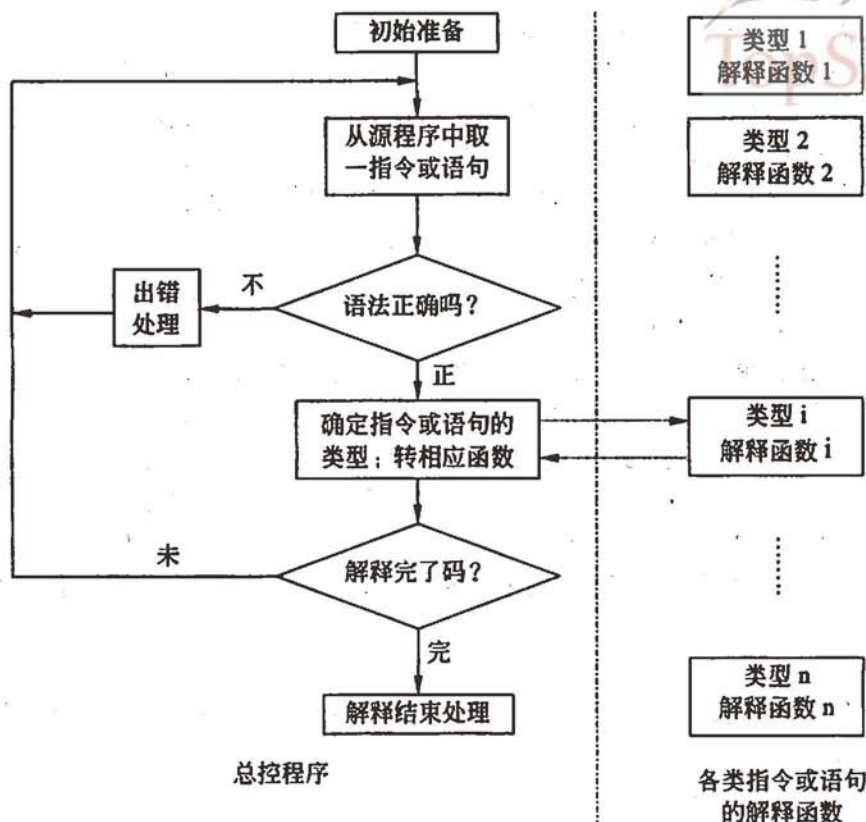


图 6-2 解释程序的结构和工作流程示意图

一般来说，采用解释方式比采用编译方式的效率低，原因是解释程序处理循环结构时，分析很复杂，对循环体的每次执行，都需要重复分析，而编译仅分析一次。

纯粹的解释和纯粹的编译是两个极端情况，很少使用它们。例如，在处理汇编语言时才使用纯粹的编译方式；在处理操作系统的控制语言和交互语言时才使用纯粹的解释方式。程序语言的通常实现方式如图 6-3 所示，是编译技术和解释技术的结合。原因是，一般来说，程序语言的执行模型与通用计算机的计算模型是不一致的。

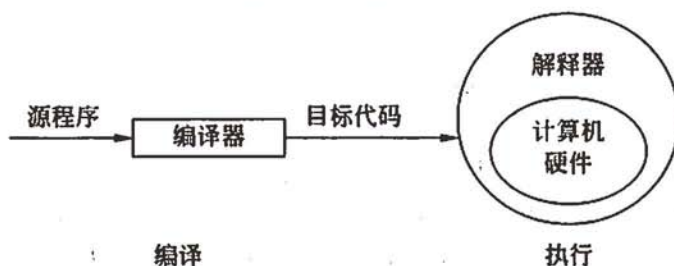


图 6-3 通常的语言实现示意图

采用哪一种处理方式,是由被实现的语言和实现环境(在什么计算机系统上实现)两者决定的,而被实现的语言,它的数据结构和控制结构更起着重要作用。程序语言被分为编译型和解释型,为了提高效率并尽早检查出源程序中的错误,尽量采用编译方式,即使采用解释方式,也尽量生成接近目标计算机代码的中间代码。

C、C++、FORTRAN、PASCAL 和 ADA 通常采用编译方式实现,称为编译型语言。编译器把源程序翻译成目标计算机语言程序,解释器只提供一个运行库,以支持目标语言程序中计算机语言没有提供的操作。一般来说,编译器比较复杂庞大,它的侧重点是产生尽可能高效运行的目标语言程序。

LISP、ML、PROLOG 和 SMALLTALK 通常采用解释方式实现,称为解释型语言。在实现中,解释器仅产生易于解释的中间代码(之所以称为中间代码,是因为它的形式很像编译器产生的中间代码形式),这种中间代码不能在硬件机上直接执行,而由解释器解释执行。这种实现比编译器相对简单,实现的复杂工作在于构造解释器。

Java 不像 LISP,而更像 C++,但由于 Java 运行在网络环境上,Java 通常被当作解释型语言,编译器产生一种字节码的中间语言,被各个终端上的浏览器创建的解釋器解释执行。

并不是任何一种汇编语言的每一语句都能对应于一条机器语言指令。一般来说,汇编语言中有三类语句:指令语句、伪指令语句和宏指令语句。伪指令语句经汇编后不产生机器语言指令。

正确答案:

(1) × (2) ✓ (3) × (4) ✓ (5) × (6) × (7) ✓ (8) ×

【例 6-7】编译程序对源程序的加工一般可分成词法分析, (1), (2), (3), (4) 和“目标代码生成”等六个阶段,其中词法分析, (1), (2) 和“目标代码生成”四个阶段是每个编译程序必不可少的,而 (3) 和 (4) 则是可有可无的,许多编译程序将词法分析设计成一个子程序,在 (1) 的分析过程中根据需要调用它,并且把词法分析, (1), (2) 和 (3) 组成一遍扫描完成。此外,各个阶段在工作过程中都会涉及到符号表管理和 (5)。

可选答案:

(1) ~ (5): A. 结构分析      B. 参数分析      C. 数据分析      D. 语法分析  
E. 过程分析      F. 出错处理      G. 代码优化      H. 表达式处理  
I. 中间代码生成      J. 语义分析

分析:

编译系统一般由编译程序、运行程序、编辑程序、调试程序等组成。编译程序对源程序的加工一般经历词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成六个阶段。从逻辑上可用图 6-4 表示。在实际的编译程序中,把某几个阶段的工作组合成一遍扫描完成。



图 6-4 编译过程的各个阶段

有的编译程序，不要求产生非常高的目标代码，可能不生成中间代码，当然也就没有对中间代码实施优化。而编译程序由若干遍扫描程序组成，每遍扫描程序完成上述六个阶段中的一个阶段或几个阶段的工作，也有的编译程序中几遍扫描程序完成一个阶段的工作。每遍扫描程序中都有表格管理程序和出错处理程序。

正确答案：

(1) D (2) J (3) I (4) G (5) F

### 6.2.3 构造编译程序基本知识

【例 6-8】有限状态自动机可用五元组  $(\Sigma, Q, \delta, q_0, Q_f)$  来描述，设有一有限状态自动机  $M$  的定义如下：

$\Sigma = \{0, 1\}$ ,  $Q = \{q_0, q_1, q_2\}$ ,  $Q_f = \{q_2\}$ ,  $\delta$  定义为：

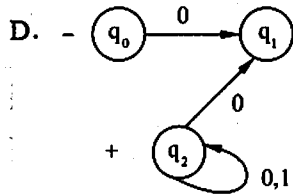
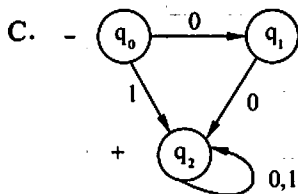
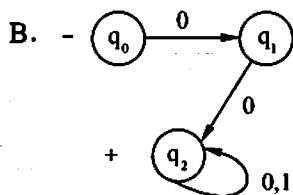
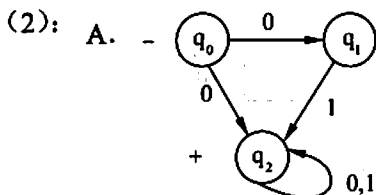
$\delta(q_0, 0) = q_1$        $\delta(q_1, 0) = q_2$   
 $\delta(q_2, 1) = q_2$        $\delta(q_2, 0) = q_2$

$M$  是一个 (1) 有限状态自动机，它所对应的状态转换图为 (2)，它所能接受的语言可以用正则表达式表示为 (3)，该正则表达式所表示的语言陈述为 (4)。

可选答案：

(1): A. 歧义的      B. 非歧义的      C. 确定的      D. 非确定的





注：图中-表示开始状态，+表示终止状态。

(3): A.  $(0|1)^*$  B.  $00(0|1)^*$  C.  $(0|1)^*00$  D.  $0(0|1)^*0$

(4): A. 由 0 和 1 所组成的符号串的集合

B. 以 0 为头符号和尾符号、由 0 和 1 所组成的符号串的集合

C. 以两个 0 结束的，由 0 和 1 所组成的符号串的集合

D. 以两个 0 开始的，由 0 和 1 所组成的符号串的集合

分析：

有限自动机可以作为一种识别装置，它能准确地识别正规集。有限自动机分为两类：确定的有限自动机和不确定的有限自动机。确定的有限自动机（DFA）定义如下：

一个确定的有限自动机（DFA） $M$  是一个五元组：

$$M = (\Sigma, Q, \delta, q_0, Q_f)$$

其中， $Q$  是一个有限状态集； $\Sigma$  是一个有穷字母表，它的每个元素称为一个输入字符；

$\delta$  是转换函数，是  $Q \times \Sigma \rightarrow Q$  的单值部分映射，如  $\delta(q, a) = q'$  ( $q, q' \in Q, a \in \Sigma$ )，意味着，在当前状态  $q$  下，输入  $a$ ，状态将转移到  $q'$ ，称  $q'$  为  $q$  的一个后继状态。

$q_0 \in Q$ ，是惟一的一个初态；

$Q_f \subseteq Q$ ，是一个终态集（可空）。

确定有限自动机的确定性表现在映射  $\delta : Q \times \Sigma \rightarrow q$  是单值函数，也就是说，对任何状态  $q \in Q$  和输入字符  $a \in \Sigma$ ， $\delta(q, a)$  惟一确定下一个状态。显然，本题给出的是一个确定有限自动机，它的状态转换图是 (2) 中的 B。

一个 DFA 可以用一个状态图表示（或称状态转换图），也可以用一个状态转换矩阵表示。例如，本题的确定有限自动机的状态转换矩阵由表 6-1 给出。

对于  $\Sigma$  上的任何字  $w$ ，若在 DFA  $M$  的状态图中存在一条从初态到终态的路径，且这条路径上的所有弧的标记符连接成的字等于  $w$ 。则称  $w$  可被 DFA  $M$  所识别（或接受）。

若 DFA  $M$  的初态又是终态, 则  $M$  能识别  $\epsilon$ , DFA  $M$  所识别的字的全体记为  $L(M)$ 。因此, 本题中的 DFA  $M$  的  $L(M)$  是开始为 00, 后面跟随  $\{0,1\}$  上的一个任意串组成的集合。

表 6-1 例 6-8 的 DFA  $M$  的状态转移矩阵

状态 \ 字符	0	1
$q_0$	$q_1$	$\emptyset$
$q_1$	$q_2$	$\emptyset$
$q_2$	$q_2$	$q_2$

用正规表达式 (简称正规式), 可描述程序语言的单词, 它表示的集合称为正规集。对于字母表  $\Sigma$  而言, 正规式和它所表示的正规集的递归定义为:

- (1)  $\epsilon$  和  $\emptyset$  是正规式, 它们所表示的正规集分别为  $\{\epsilon\}$  和  $\emptyset$ ;
- (2)  $\forall a \in \Sigma$ ,  $a$  是正规式, 它所表示的正规集为  $\{a\}$ ;
- (3) 设  $r$  和  $s$  是  $\Sigma$  上的正规式, 它们所表示的正规集分别为  $L(r)$  和  $L(s)$ , 那么,  $(rs)$ 、 $(rs)$ (连结) 和  $(r)$  也是正规式, 它们所表示的正规集分别为  $L(r) \cup L(s)$ 、 $L(r)L(s)$  和  $L(r)$ 。
- (4) 仅由有限次使用以上三个运算构造出的表达式才是正规式。

正规式中运算符的优先级从高到低依次是:  $*$ 、连结和  $|$ 。

算术表达式表示计算规则, 已知一个算术表达式, 人们按照它所表示的计算规则, 能计算出一个算术值; 正则表达式表示集合的计算规则, 已知一个正则表达式, 按照它所表示的计算规则, 通过计算能得到一个正规集合。

仿照计算算术表达式计算正则表达式  $00(0|1)^*$  如下:

$$\begin{aligned}
 & 00(0|1)^* && /* \quad 00 \text{ 表示为 } \{0\}\{0\} \quad */ \\
 & = \{0\}\{0\}(\{0\} \cup \{1\})^* && /* \quad 0|1 \text{ 表示为 } \{0\} \cup \{1\} \quad */ \\
 & = \{00\}\{0, 1\}^* \\
 & = \{00\}\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\} \\
 & = \{00, 000, 001, 0000, 0001, 0010, 0011, 00000, 00001, \dots\}
 \end{aligned}$$

因此, 正则表达式  $00(0|1)^*$  表示的集合为由两个 0 开始的后跟由 0 和 1 组成的任一符号串构成的集合。

正确答案:

- (1) C (2) B (3) B (4) D

【例 6-9】已知一个不确定的有穷自动机 (NFA) 如图 6-5 所示, 采用子集法将其确定化为 DFA 的过程如表 6-2 所示。

状态集  $T_1$  中不包括编号为 (1) 的状态; 状态集  $T_2$  中的成员有 (2); 状态集  $T_3$  等于 (3); 该自动机所识别的语言可以用正规式 (4) 表示。与正规式  $(a|b)^*$  等价的正规式为 (5)。

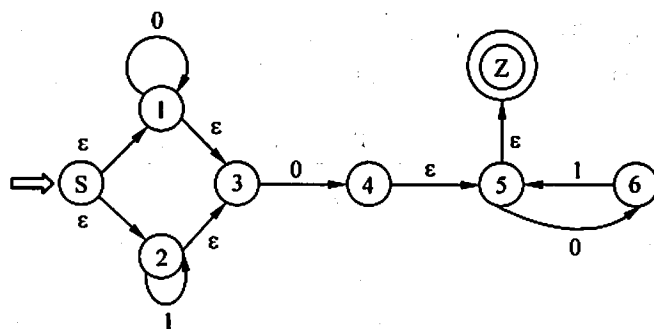


图 6-5 一个非确定的有限自动机 (NFA)

表 6-2 图 6-5 所示 NFA 的确定性过程

I	$I_0$	$I_1$
{S, 1, 2, 3}	{1, 3, 4, 5, Z}	{2, 3}
{1, 3, 4, 5, Z}	T1	T3
{2, 3}	{4, 5, Z}	{2, 3}
T2	{6}	T3
T1	{1, 3, 4, 5, 6, Z}	{5, Z}
{6}	T3	{5, Z}
{5, Z}	{6}	T3

可选答案:

- (1): A. 2                      B. 4                      C. 3                      D. 5
- (2): A. 13,4,5,Z              B. 2,3                      C. 6                      D. 4,5,Z
- (3): A. {Z}                      B. {6}                      C. {4,5,Z}              D. {}
- (4): A.  $(0|1)^*$                       B.  $(0^*|1^*)^*001$   
       C.  $(0^*|1^*)^*0(0|1)^*$                       D.  $(0^*|1^*)0(0|1)^*$
- (5): A.  $a^*|b^*$                       B.  $a^*b^*$                       C.  $(a^*b^*)^*$                       D.  $(ab)^*$

分析:

一个非确定的有限自动机 (NFA)  $M$  是一个五元组:

$$M = (\Sigma, Q, \delta, q_0, Q_f)$$

其中,  $\Sigma, Q, q_0, Q_f$  的含义同确定的有限自动机,  $\delta$  和确定的有限自动机不同, 其含义是:  $\delta$  是一个从  $Q \times \Sigma^*$  到  $Q$  子集的映射, 即

$$\delta : Q \times \Sigma^* \rightarrow 2^Q$$

和确定的有限自动机一样, 可以定义一个 NFA  $M$  所能识别 (或接受) 的字。

显然, DFA 是 NFA 的特例。但是, 对于每个 NFA  $M$  都存在一个 DFA  $M'$ , 使得  $L(M') = L(M)$ 。

有一个方法称为子集构造法, 能将一个非确定的有限自动机转换成一个等价的确定的有限自动机。

具体说来, 对于给定的一个 NFA  $M$ , 我们设想有一个 DFA  $M'$ , 它的初态是 NFA  $M$  的初态  $q_0$  以及从  $q_0$  出发沿空弧所能到达的那些状态, 表示成  $I = \varepsilon\_closure(q_0)$ 。在  $M$  中一个状态和一个输入符号可能转换到多个状态, 若在 NFA  $M$  中有  $I \times a \in \Sigma \rightarrow J \subseteq Q$  (表示成  $J = move(I, a)$ ), 那么, 在 DFA  $M'$  中设状态  $I_a = \varepsilon\_closure(J)$ ,  $\varepsilon\_closure(J)$  称为  $\varepsilon$ -闭包, 其计算方法下面予以介绍。这实际上是用 DFA  $M'$  模拟 NFA  $M$  的动作, 重复这个模拟过程, 直到  $M'$  中不再增加新的状态。这个过程, 将逐步构造出 DFA  $M'$  的状态转移矩阵如表 6-2 所示。

$\varepsilon\_closure(T)$  称为子集  $T$  的  $\varepsilon$ -闭包, 计算方法如下:

- (1)  $\varepsilon\_closure(T) = T$ ;
- (2)  $\forall q \in \varepsilon\_closure(T)$ , 若  $\delta(q, \varepsilon) = q'$ , 则把  $q'$  加到  $\varepsilon\_closure(T)$  中, 直到  $\varepsilon\_closure(T)$  不再增大为止。

也就是说,  $\varepsilon\_closure(T)$  不仅含有  $T$ , 而且含有从  $T$  出发沿空弧所能到达的所有状态。直观理解是去掉 NFA  $M$  的空弧。

表 6-2 中  $I$  是  $M$  状态集的一个子集, 首先求 DFA  $M'$  的初态,

表 6-2[0, 0] =  $\varepsilon\_closure(s) = \{s, 1, 2, 3\}$ , 然后求  $I_0$  和  $I_1$ 。

$T1 = \varepsilon\_closure(move(\{1, 3, 4, 5, z\}, 0))$

$= \varepsilon\_closure(\{1, 4, 6\}) = \{1, 3, 4, 5, 6, z\}$

$T3 = \varepsilon\_closure(move(\{1, 3, 4, 5, z\}, 1)) = \varepsilon\_closure(\{\}) = \{\}$

如果  $I_0$  和  $I_1$  不出现在表的第一列  $I$  中, 则把它们填入第一列  $I$  的下面的空行位置上。之后, 对  $I$  的新行上的子集重复上述过程, 直至所有第二列和第三列的子集全都在第一列中出现为止。这个过程必定在有限步内终止, 因为  $M$  的状态子集的个数是有限的。

按下面图 6-6 所示的方法逐步消去 NFA  $M$  中除初态结  $s$  和终态结  $z$  的所有结点。在消除结点的过程中, 用正规式来标记弧, 最后初态结  $s$  和终态结  $z$  之间的弧上的标记就是所求的正规式。

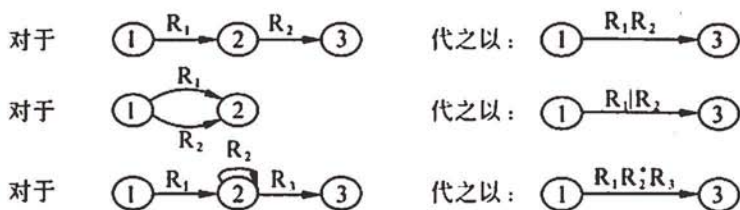
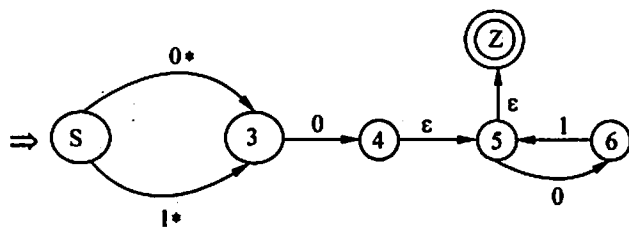


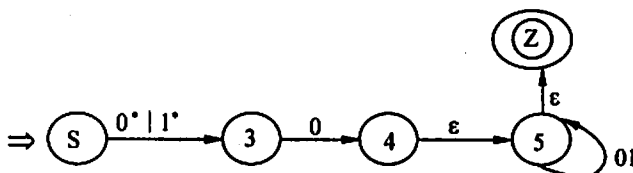
图 6-6 有限自动机到正规式的转换规则示意图

图 6-5 所示 NFA  $M$  到正规式的转换过程如图 6-7 所示。

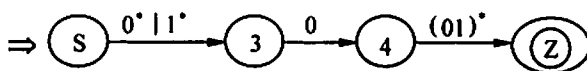




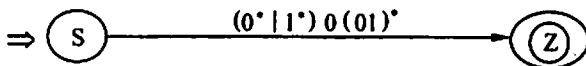
(a) 削除状态结 1 和 2



(b) 合并结 s 和结 3 之间的两条弧且削除状态结 6



(c) 削除状态结 5



(d) 削除状态结 3 和 4

图 6-7 图 6-5 所示 NFA M 到正规式的转换过程

下面说明  $(a|b)^*$  和  $(a^*b^*)^*$  等价。证明两个正规式等价的方法不止一种，下面的方法是说明  $L((a|b)^*) = L((a^*b^*)^*)$ 。

因为  $L(a|b) \subseteq L(a^*b^*)$  所以  $L((a|b)^*) \subseteq L((a^*b^*)^*)$ 。

又因为  $L(a^*) \subseteq L((a|b)^*)$ ,  $L(b^*) \subseteq L((a|b)^*)$ ,

所以  $L(a^*b^*) \subseteq L((a|b)^*(a|b)^*) = L(((a|b)^*)^2) = L((a|b)^*)$ ,

$L((a^*b^*)^*) \subseteq L(((a|b)^*)^*) = L((a|b)^*)$  /\*  $r^* = (r^*)^*$  \*/

最后,  $(a|b)^*$  和  $(a^*b^*)^*$  等价。

正确答案:

(1) A (2) D (3) D (4) D (5) C

【例 6-10】描述程序设计语言语法的 BNF 表示法中, “ $\rightarrow$ ” (有时用 “ $::=$ ” 表示) 表示 (1), “ $|$ ” 表示 (2), 为了方便, 通常引入如下表示,  $[W]$  表示  $W$  出现 (3) 次,  $\{W\}$  表示  $W$  出现 (4) 次。

设某种语言的 ON 语句的语法规则如下:

$\langle \text{ON 语句} \rangle \rightarrow \text{ON} \langle \text{变量} \rangle [\text{GOTO} \langle \text{标号} \rangle \{, \langle \text{标号} \rangle \}$

$\langle \text{变量} \rangle \rightarrow A|B|\dots|Z$

<标号>→L1|L2|...|L9

则在供选择的答案中,不符合语法的语句是 (5)。

可选答案:

- (1): A. 恒等于      B. 不等于      C. 取决于      D. 定义为  
 (2): A. 与      B. 或      C. 非      D. 引导开关参数  
 (3), (4): A. 1      B.  $n (n \geq 0)$       C.  $n (n \geq 1)$       D. 0 或 1  
 (5): A. ON A GOTO L1      B. ON B L1,L2,L3  
       C. ON Z GOTO L1 L2      D. ON C L2,L3

分析:

语言是句子的集合,而句子是根据文法中的规则推导出来的。

(1) 上下文无关文法

描述语言的语法结构的形式规则称为文法。一个上下文无关文法(简称文法)  $G$  是一个四元组,可表示为

$$G = (V_T, V_N, S, P)$$

其中,  $V_T$  是一个非空有限集合,每个元素称为终结符号。 $V_N$  是一个非空有限集合,每个元素称为非终结符号。 $V_T \cap V_N = \emptyset$ 。用  $V$  表示  $V_T \cup V_N$ , 称  $V$  为文法  $G$  的文法符号集合。 $S \in V_N$ , 称为开始符号,它至少要在一条产生式中作为左部出现一次。 $P$  是一个有限的具有如下形式的产生式(也称重写规则)集合:

$$A \rightarrow \beta$$

其中:  $A \in V_N$ , 称为产生式的左部;  $\beta \in V^*$ , 称为产生的右部。若干左部相同的产生式  $A \rightarrow \beta_1, A \rightarrow \beta_2, \dots, A \rightarrow \beta_n$ , 可简写为  $A \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$ ,  $\beta_i (1 \leq i \leq n)$  为  $A$  的一个候选式。

" $A \rightarrow \beta$ " 的含义是  $A$  定义为  $\beta$ , 或  $A$  由  $\beta$  组成。

" $|$ " 的含义是 "或者"。

(2) 推导与归约

设文法  $G = (V_T, V_N, S, P)$ ,  $A \rightarrow \beta \in P$ ,  $\gamma, \delta \in V^*$ , 则称  $\gamma A \delta$  直接推导出  $\gamma \beta \delta$ , 表示成

$$\gamma A \delta \Rightarrow \gamma \beta \delta$$

也称  $\gamma \beta \delta$  直接归约到  $\gamma A \delta$ 。

这个定义告诉我们,若知道  $\gamma A \delta \in V^*$ , 根据  $A \rightarrow \beta \in P$ , 可求出  $\gamma \beta \delta \in V^*$ ; 反过来, 知道  $\gamma \beta \delta \in V^*$ , 根据  $A \rightarrow \beta \in P$ , 可求出  $\gamma A \delta \in V^*$ 。

若存在一个推导序列:  $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n (n > 0)$ , 则称从  $\alpha_0$  经  $n$  步推导出  $\alpha_n$ , 表示成  $\alpha_0 \xRightarrow{+} \alpha_n$ 。若  $\alpha_0 = \alpha_n$  或  $\alpha_0 \xRightarrow{+} \alpha_n$ , 则表示成  $\alpha_0 \xRightarrow{*} \alpha_n$ 。

(3) 句子和语言

若  $G = (V_T, V_N, S, P)$  且  $S \xRightarrow{*} w$ , 则称  $w$  是文法  $G$  的一个句型。仅含终结符号的句型是一个句子。语言  $L(G)$  是由文法  $G$  产生的所有句子组成的集合:

$$L(G) = \{w | S \xRightarrow{+} w \text{ 且 } w \in V_T^*\}$$

[w]表示或者有 w, 或者没有 w; {w}表示  $w^*$ 。

根据文法的第一条规则, ON 语句的标号表中标号之间是用逗号隔开的, 而 (5) 中的 C 的 L1 和 L2 之间没有逗号。

正确答案:

(1) D (2) B (3) D (4) B (5) C

【例 6-11】已知,  $G_0 = (\{a, b\}, \{S, X, Y\}, P, S)$ , P 中的产生式及序号如下:

①:  $S \rightarrow XaaY$

②:  $X \rightarrow YY|b$

③:  $Y \rightarrow XbX|a$

则  $G_0$  为 (1) 型文法, 对应于 (2), 由  $G_0$  推导出句子 aaaaa 和 baabbb 时, 所使用的产生式序号组成的序列分别为 (3), (4)。

文法  $G_1[S]: S \rightarrow xSx|y$  所描述的语言是 (5) ( $n \geq 0$ )。

可选答案:

(1): A. 0 B. 1 C. 2 D. 3

(2): A. 图灵机 B. 下推自动机

C. 有限状态自动机 D. 其他自动机

(3), (4): A. ①③①③③ B. ①②③①②

C. ①②③②② D. ①②③③③

(5): A.  $x^n y$  B.  $x^n y x^m$  C.  $x^n y^n$  D.  $x^n y x^n$

分析:

乔姆斯基 (Chomsky) 把文法分成四种类型, 即 0 型、1 型、2 型和 3 型, 由之产生的语言分别称为 0 型、1 型、2 型和 3 型语言。这几类文法的差别在于对产生式的形式施加不同的限制, 如表 6-3 所示。

表 6-3 文法的类型

文法的类型	产生式形式	文法产生的语言
0	$\alpha \rightarrow \beta$ , 其中 $\alpha, \beta \in V^*$ , $\alpha \neq \epsilon$	0
1 (上下文有关)	$\alpha \rightarrow \beta$ , 其中 $\alpha, \beta \in V^*$ , $ \alpha  \leq  \beta $	1
2 (上下文无关)	$A \rightarrow \beta$ , 其中 $A \in V_N, \beta \in V^*$	2 (上下文无关语言)
3 (正规)	$A \rightarrow a$ 或 $A \rightarrow aB$ (右线性) $A \rightarrow a$ 或 $A \rightarrow Ba$ (左线性) 其中 $A, B \in V_N, a \in V_T \cup \{\epsilon\}$	3 (正规语言)

0 型文法也称短语文法, 其能力相当于图灵机。1 型文法也称上下文有关文法, 其能力相当于线性界限自动机。2 型文法也称上下文无关文法, 其能力相当于非确定的下推自动机。3 型文法也称线性文法 (或称正规文法), 其能力相当于有限自动机。

$G_0$  的所有产生式形式都是  $A \rightarrow \beta$ , 其中  $A \in V_N$ ,  $\beta \in V^*$ , 属于 2 型文法, 其能力相当于确定的下推自动机。

(3) 和 (4) 的推导过程如下:

(3):  $S \Rightarrow XaaY$  (使用产生式的序号①)

$\Rightarrow YYaaY$  (使用产生式的序号②)

$\Rightarrow aYaaY$  (使用产生式的序号③)

$\Rightarrow aaaaY$  (使用产生式的序号③)

$\Rightarrow aaaaa$  (使用产生式的序号③)

(4):  $S \Rightarrow XaaY$  (使用产生式的序号①)

$\Rightarrow baaY$  (使用产生式的序号②)

$\Rightarrow baaXbX$  (使用产生式的序号③)

$\Rightarrow baabbX$  (使用产生式的序号②)

$\Rightarrow baabbb$  (使用产生式的序号②)

故 (3) 应选择 D, (4) 应选择 C。

文法  $G_1[S]$  的产生式是  $S \rightarrow xSx|y$ , 生成句子或者用  $S \rightarrow y$  经一步推导产生, 或者用  $S \rightarrow xSx$  推导若干步, 最后一步推导用  $S \rightarrow y$  产生:

$S \Rightarrow y$

$S \Rightarrow xSx \Rightarrow xxSxx = x^2Sx^2 \Rightarrow x^2xSx^2x = x^3Sx^3 \Rightarrow \dots \Rightarrow x^nSx^n \Rightarrow x^nyx^n$

正确答案:

(1) C (2) B (3) D (4) C (5) D

**【例 6-12】** 语法分析分自顶向下分析和自底向上分析。自顶向下分析试图为输入串构造一个 (1), 画分析树是从 (2) 画向 (3), 需要解决的关键问题是根据当前读入符号和被替换的非终结符号, 如何选择这个非终结符号的 (4) 进行推导; 自底向上分析试图为输入串构造一个 (5), 画分析树是从 (6) 画向 (7), 需要解决的关键问题是如何找出右句型的 (8)。

可选答案:

(1), (5): A. 分析树 B. 推导 C. 最左推导 D. 最右推导

(2), (3), (6), (7): A. 上 B. 下 C. 树根 D. 树叶

(4), (8): A. 终结符 B. 非终结符 C. 候选式 D. 句柄

分析:

设语法分析从左至右读入串  $w$ , 自顶向下分析是对于符合文法的输入  $w$ , 构造一个最左推导; 自底向上分析是对于符合文法的输入  $w$ , 按最右推导的逆过程构造一个最右推导。对于违反文法规定的输入, 指出其中的错误。关键问题是, 面对当前输入, 选择哪条产生式进行推导。

在构造最左推导的过程中, 面对当前读入的单词符号和当前被替换的非终结符两



者, 应该选择这个非终结符的哪条候选式去替换它(推导); 而在构造最右推导的过程中, 面对当前读入的单词符号, 已分析过的符号串是否已构成一个产生式的右部, 即句柄。如果已构成句柄, 即用相应的产生式左部(非终结符号)去替换它(归约)。因此, 对于自顶向下分析, 主要找出选择一个非终结符号的候选式的方法; 而对于自底向上分析, 主要讨论寻找句柄的方法。

用分析树图示自顶向下构造最左推导的过程, 开始从根结点出发, 随着分析的步步进展, 分析树节节生长, 最后长出树叶。

用分析树图示自底向上构造最右推导的过程, 请注意它是按最右推导的逆过程构造最右推导的, 和人们读书一样, 边读边分析边归约, 因此, 画分析树是从树叶开始, 画向树根。

对于自顶向下分析, 预测分析要求文法满足一定的条件, 读者应掌握递归子程序方法和 LL(1) 分析方法; 对于自底向上的分析, 读者应掌握归约和句柄的概念以及算符优先分析法和 LR 分析法。

正确答案:

(1) C (2) C (3) D (4) C (5) D (6) D (7) C (8) D

【例 6-13】 已知文法  $G[S]$  的产生式如下:

$$S \rightarrow (L) \mid a$$

$$L \rightarrow L, S \mid S$$

属于  $L(G[S])$  的句子是 (1),  $(a,a)$  是  $L(G[S])$  的句子, 这个句子的最左推导是 (2), 最右推导是 (3), 分析树是 (4), 句柄是 (5)。

可选答案:

(1): A. a B. a,a C. (L) D. (L,a)

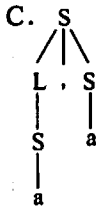
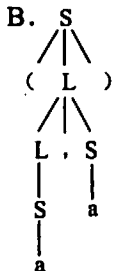
(2), (3):

A.  $S \Rightarrow (L) \Rightarrow (L,S) \Rightarrow (L,a) \Rightarrow (S,a) \Rightarrow (a,a)$

B.  $S \Rightarrow (L) \Rightarrow (L,S) \Rightarrow (S,S) \Rightarrow (S,a) \Rightarrow (a,a)$

C.  $S \Rightarrow (L) \Rightarrow (L,S) \Rightarrow (S,S) \Rightarrow (a,S) \Rightarrow (a,a)$

(4): A.



(5): A.  $(a,a)$  B. a,a C. (a) D. a

分析:

设文法  $G = (V_T, V_N, S, P)$ ,  $A \rightarrow \beta \in P$ ,  $\gamma, \delta \in V^*$ , 则称  $\gamma A \delta$  直接推导出  $\gamma \beta \delta$ , 表示成

$$\gamma A \delta \Rightarrow \gamma \beta \delta \quad (6-1)$$

也称  $\gamma \beta \delta$  直接归约到  $\gamma A \delta$ 。

对于 (6-1), 若  $\gamma \in V_T^*$ , 即  $A$  是  $\gamma A \delta$  中最左边的非终结符号, 则称 (6-1) 是一个最左推导。若  $S \xRightarrow{+} \alpha$  的每一步都是最左推导, 则称  $S \xRightarrow{+} \alpha$  是一个最左推导,  $\alpha$  称为一个左句型。

对于 (6-1), 若  $\delta \in V_T^*$ , 即  $A$  是  $\gamma A \delta$  中最右边的非终结符号, 则称 (6-1) 是一个最右推导。若  $S \xRightarrow{+} \alpha$  的每一步都是最右推导, 则称  $S \xRightarrow{*} \alpha$  是一个最右推导,  $\alpha$  称为一个右句型。最右推导也称做规范推导, 右句型也称做规范句型。

对于句子  $(a, a)$ , 供选择的答案的 (2) 和 (3) 中,  $C$  是最左推导,  $A$  是最右推导。供选择的答案的 (4) 中的  $B$  是它的分析树。

在自底向上分析的过程中, 按最右推导的逆过程构造出最右推导, 称为规范规约。关键是每步找出被归约的右句型的“可归约串”, 称为“句柄”。请读者仔细领会句柄的定义:

右句型 (最右推导推导出的句型)  $\gamma$  的句柄是一个产生式  $A \rightarrow \beta$  以及  $\gamma$  中的一个位置, 根据这个位置可找到  $\beta$ , 用  $A$  代替  $\beta$  得到最右推导的前一个右句型。即, 如果有下面的最右推导:

$$S \xRightarrow{+} \alpha A w \Rightarrow \alpha \beta w$$

那么, 在  $\alpha$  后和  $A \rightarrow \beta$  是  $\alpha \beta w$  的句柄。句柄右边的  $w$  仅含终结符号。

有的教课书上, 句柄的定义借助于短语、直接短语的定义给出:

设  $G = (V_T, V_N, S, P)$  是一个文法, 若

$$S \xRightarrow{*} \alpha A \gamma \xRightarrow{+} \alpha \beta \gamma$$

则在句型  $\alpha \beta \gamma$  中  $\beta$  是相对于非终结符号  $A$  的短语。又若

$$S \xRightarrow{*} \alpha A \gamma \Rightarrow \alpha \beta \gamma$$

则在句型  $\alpha \beta \gamma$  中  $\beta$  是相对于非终结符号  $A$  的直接短语。最左边的直接短语称为句柄。

根据句子  $(a, a)$  的最右推导:

$$S \xRightarrow{+} (S, a) \Rightarrow (a, a) \quad (\text{注: 此步最右推导使用规则 } S \rightarrow a)$$

因此,  $(a, a)$  中左边的  $a$  是句型  $(a, a)$  的句柄。在分析树中, 最左最下边的子树的树叶从左至右构成的串是句柄。请看  $(a, a)$  的分析树。

正确答案:

(1) A (2) C (3) A (4) B (5) D

**【例 6-14】** 自底向上分析方法采用自左向右扫描输入符号串, 通过 (1) 分析其语法是否正确。LR 分析法属于自底向上分析方法, 一个 LR 分析器的逻辑结构由一个输入,

一个输出, 一个栈, 一个 LR 驱动程序, 一个 (2) 组成。LR 分析主要有 SLR (1)、LR (0)、LR (1)、LALR (1) 四种, 而它们的 LR 驱动程序是 (3) 的。LR 分析的动作由当前状态和当前读入符号两者决定, 动作有移进, (4), (5), 出错处理四种。

可选答案:

- (1): A. 归约—移进                      B. 移进—移进  
       C. 移进—归约                      D. 归约—归约
- (2): A. 优先表            B. 调度表            C. 操作数表            D. 分析表
- (3): A. 完全不一样                      B. 基本不一样  
       C. 大致一样                        D. 完全一样
- (4), (5): A. 匹配    B. 归约            C. 接受                D. 进栈

分析:

LR 分析是一种自底向上分析方法, 具体实现采用移进—归约分析法。请看例子, 已知文法  $G = (\{a, b, c, d, e\}, \{S, A, B\}, S, P)$ , 其中  $P$  为:

$S \rightarrow aABe, A \rightarrow Abc \mid b, B \rightarrow d$

句子  $abbcd$  的最右推导:

$S \Rightarrow aABe \Rightarrow aAde \Rightarrow aAbcde \Rightarrow abbcde$

LR 分析按最右推导的逆过程:

$abbcde \Leftarrow aAbcde \Leftarrow aAde \Leftarrow aABe \Leftarrow S$

构造出最右推导。具体的分析过程如表 6-4 所示。

最右推导称规范推导, 其逆过程称为规范归约。若用分析树图示出来, 它是从树叶开始, 一步步画向树根, 因此称自底向上分析。

一个 LR 分析器的逻辑结构如图 6-8 所示, 由五部分组成: 一个输入, 一个输出, 一个栈, 一个 LR 驱动程序和一个分析表。

表 6-4 分析句子  $abbcde$  的移进—归约过程

栈	输入字符串	动作	栈	输入字符串	动作
#	abbcde#	移进	#aA	de#	移进
#a	bbcd#	移进	#aAd	e#	归约 ( $B \rightarrow d$ )
#ab	bcde#	归约 ( $A \rightarrow b$ )	#aAB	e#	移进
#aA	bcde#	移进	#aABe	#	归约 ( $S \rightarrow aABe$ )
#aAb	cde#	移进	#S	#	接受
#aAbc	de#	归约 ( $A \rightarrow Abc$ )			

① LR 驱动程序: 所有 LR 分析器的驱动程序都是相同的。

② 分析表: 不同的文法具有不同的分析表。同一文法采用不同的 LR 分析方法分析时, 分析表也不同。分析表又可分为动作表 (ACTION) 和状态转换表 (GOTO) 两个

部分, 它们都可用二维数组表示。

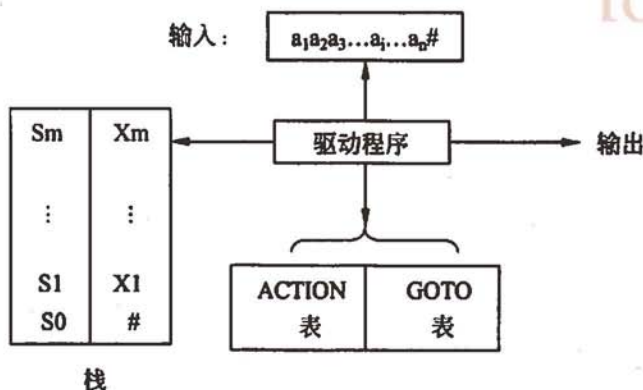


图 6-8 一个 LR 分析器的模型

③ 分析栈：包括文法符号栈和相应的状态栈。

LR 驱动程序根据栈顶状态和当前输入符号查分析表以决定分析器下一步的动作。状态转换表用  $GOTO[S_m, X]=S_j$  表示, 规定当栈顶状态为  $S_m$ , 遇到当前文法符号  $X$  时应转向状态  $S_j$ 。ACTION[ $S_m, a$ ] 规定了栈顶状态为  $S_m$ , 遇到输入符号  $a$  时应执行的动作, 有以下四种可能。

① 移进: 把  $S_j = GOTO[S_m, a]$  移进状态栈,  $a$  移进文法符号栈, 其中  $i, j$  表示状态号。

② 归约: 此时栈顶已形成句柄, 则把它归约为相应产生式的左部。

③ 接受 acc: 此时文法符号栈中只剩文法的开始符号  $S$ , 并且当前输入符是 “#”, 分析成功。

④ 报错: 此时在状态栈顶状态下出现不该遇到的输入符号, 说明输入串不是该文法所能接受的句子。

设计 LR 分析器的关键工作是构造 LR 分析表, 在此不作介绍, 请阅读有关资料。

正确答案:

(1) C (2) D (3) D (4) B (5) C

【例 6-15】已知文法  $G[S]$  的产生式如下:

$$S \rightarrow (L) \mid a \quad L \rightarrow L, S \mid S$$

其相应的 LL (1) 分析表如下:

	a	,	(	)	#
S	$S \rightarrow a$		(3)		
L	(1)		(4)		
M		(2)		(5)	





号是 A, 当前输入符号在  $FIRST(\alpha)$  中时, 显然应选择候选式  $\alpha$  替换 A, 即在分析表中的项  $M[A, a]$  应填产生式  $A \rightarrow \alpha$ 。何时填  $A \rightarrow \epsilon$ , 显然 a 应是跟随 A 的终结符号, 为此定义 FOLLOW(A) 如下:

$$FOLLOW(A) = \{a \mid S \xRightarrow{*} \dots Aa \dots, a \in V_T, A \in V_N\}$$

若  $S \xRightarrow{*} \dots A$ , 则规定  $\# \in FOLLOW(A)$ 。

给定一个文法, 求它的每个非终结符号的  $FIRST(\alpha)$ , 可直接根据定义求出; 求 FOLLOW(A) 的算法如下:

- (1) 对于文法的开始符号 S, 置  $\#$  于 FOLLOW(S) 中;
- (2) 若  $A \rightarrow \alpha B \beta \in P$ , 则把  $FIRST(\beta)$  中的所有非  $\epsilon$  元素都加至 FOLLOW(B) 中;
- (3) 若  $A \rightarrow \alpha B \in P$ , 或  $A \rightarrow \alpha B \beta \in P$  而  $\beta \xRightarrow{*} \epsilon$ , 则把 FOLLOW(A) 加至 FOLLOW(B) 中。

重复使用上述三条规则, 直到每个 FOLLOW 集合不再增大为止。

对于文法 G[E] (6-2), 其  $FIRST(\alpha)$  和 FOLLOW(A) 如表 6-5 所示:

表 6-5 文法 G[E] (6.2) 的  $FIRST(\alpha)$  和 FOLLOW(A)

	FIRST	FOLLOW
S	( a	# , )
L	( a	)
M	, ε	)

预测分析表的构造方法如下:

- (1) 对文法 G 的每条产生式  $A \rightarrow \alpha$  执行第 (2) 步和第 (3) 步;
- (2) 对  $\forall a \in FIRST(\alpha)$ , 把  $A \rightarrow \alpha$  填入中  $M[A, a]$  中;
- (3) 若  $\epsilon \in FIRST(\alpha)$ , 则对  $\forall b \in FOLLOW(A)$ , 把  $A \rightarrow \alpha$  填入  $M[A, b]$  中;
- (4) 把所有无定义的  $M[A, a]$  填上出错处理入口。

文法 G[E] (6-2) 的预测分析表如表 6-6 所示。

表 6-6 文法 G[E] (6-2) 的 LL(1) 分析表

	a	,	(	)	#
S	$S \rightarrow a$		$S \rightarrow (L)$		
L	$L \rightarrow SM$		$L \rightarrow SM$		
M		$M \rightarrow ,SM$		$M \rightarrow \epsilon$	

G[E] (6.2) 的分析表 M 不含多重定义入口, 称这样的分析表是 LL(1) 分析表, 这样的文法是 LL(1) 文法, 由此进行的分析也称 LL(1) 分析。

根据 LL(1) 分析表的构造方法, 一个文法 G 是 LL(1) 文法, 当且仅当对于文法 G 的每一个非终结符号 A 的任何两个不同的产生式  $A \rightarrow \alpha \mid \beta$ , 下面的条件成立:

- (1)  $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$ ;

(2) 若  $\beta \xrightarrow{*} \varepsilon$ , 那么,  $FIRST(\alpha) \cap FOLLOW(A) = \emptyset$ 。

正确答案:

(1) C (2) D (3) A (4) C (5) E

【例 6-16】编译程序一般使用 (1) 的方法进行语义分析和生成中间代码。采用的中间代码形式一般有后缀式、三元式、四元式、树形表示等。已知赋值语句  $x := (a-b) * (c-d)$ , 它的后缀式是 (2), 三元式是 (3), 四元式是 (4), 树形表示是 (5)。

可选答案:

(1): A. 自底向上翻译

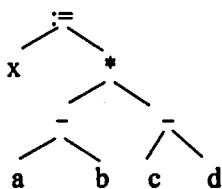
B. 自顶向下翻译

C. 协同翻译

D. 语法制导翻译

(2) ~ (5): A.  $xab-cd-*:=$

B.



C. ① (-, a, b)

② (-, c, d)

③ (\*, ①, ②)

④ (:=, ③, x)

D. ① (-, a, b, t1)

② (-, c, d, t2)

③ (\*, t1, t2, t3)

④ (:=, t3, , x)

分析:

首先, 根据语义分析和生成中间代码的需要, 给每个文法符号设置若干个属性, 用以表示它的语义信息。例如, 变量可设置类型、层次、内存单元地址等。表达式可设置类型、中间代码等。然后对语言的文法进行扩充, 对每条产生式编写一个语义子程序 (或函数), 用以计算有关文法符号的属性值, 即在语法分析过程中使用这条产生式进行语法分析时调用它的语义子程序完成相应的语义分析和翻译任务。

随着语法分析的步步进展, 当一条产生式获得匹配 (自顶向下分析) 或用于归约 (自底向上分析) 时, 就执行这条产生式所对应的语义子程序进行翻译, 这种方法称做语法制导翻译。它既适用于自顶向下分析, 又适用于自底向上分析。

从原理上, 源程序在进行了语义分析之后就可以直接生成目标代码, 但由于源程序与目标代码的逻辑结构往往差别很大, 特别是考虑到具体机器指令系统的特点, 要一遍扫描生成高效率的目标代码是困难的, 因此有必要设计一种中间代码, 将源程序首先翻译成中间代码表示形式, 以利于进行与机器无关的优化处理。由于中间代码实际上也起着编译器前端和后端分水岭的作用, 所以使用中间代码也有助于提高编译程序的可移植性。常用的中间代码有后缀式、三元式、四元式和树形表示等。

(1) 后缀式 (逆波兰形式)。逆波兰形式是波兰逻辑学家卢卡西维奇 (Lukasiewicz)

发明的一种表示表达式的方法。他把运算符写在运算对象的后面,例如,把  $a+b$  写成  $ab+$ , 所以也称为后缀式。这种表示法的优点是根据运算对象和算符的出现次序进行计算,不需要使用括号。赋值语句  $x:=(a-b)*(c-d)$  的后缀式是“ $xab-cd-*:=$ ”。

(2)  $x:=(a-b)*(c-d)$  的树形表示是(2)~(5)选择中的 B。这种树形表示称为语法树(或抽象语法树),而用以图示语法分析的树称做分析树。

(3) 三元式表示。每条三元式的形式是  $(OP, ARG1, ARG2)$ , 三元式出现的先后顺序和表达式各部分的计值顺序是一致的。

(4) 四元式表示。每条四元式的形式是  $(OP, ARG1, ARG2, RESULT)$ , 是一种经常采用的中间代码形式,其中,运算对象和运算结果有时指用户自定义的变量,有时指编译程序引入的临时变量,  $RESULT$  常常是一个新引进的临时变量,用来存放运算的结果。

三元式和四元式是三地址代码的两种实现方式,三地址代码更好读,赋值语句  $x:=(a-b)*(c-d)$  的三地址代码表示是:

```
t1:= a-b
t2:=c-d
t3:=t1*t2
x:=t3
```

正确答案:

(1) D (2) A (3) C (4) D (5) B

【例 6-17】运行时的存储分配策略在设计语言的处理系统中是非常重要的。一般来说,早先的 FORTRAN 语言的编译系统采用(1)的存储分配策略,ALGOL 类语言的编译系统采用(2)的存储分配策略,而 LISP 语言的处理系统采用(3)的存储分配策略。一个语言中不同种类的变量往往采用不同的存储分配策略,C 语言中的全局变量和静态变量采用(4),而自动类变量采用(5)。

可选答案:

(1)~(3): A. 栈式分配      B. 最佳分配      C. 堆式分配  
D. 静态分配      E. 随机分配      F. 首先分配

分析:

在目标程序运行时,目标代码所引用的数据对象在内存空间中。因此,在编译阶段产生目标时,要把目标代码所引用的数据对象映射到内存空间上,在运行时,再分配给需要的内存单元,这个过程称为存储分配。

分配的对象有简单数据类型(如整、实和布尔型等)、结构数据类型(如数组和记录等)和连接数据(如返回地址、参数等)。由于各种语言的语义不同,主要是名字的作用域和生存期不同,因而对存储空间的组织 and 分配也不同。存储分配策略可分为静态和动态两大类。



把过程或函数的一次执行叫一个活动，把一个活动所需要的存储空间组织在一起构成一个活动记录（有的书上称数据区），不同语言的活动记录结构是不同的。

如果在编译时能确定目标程序运行中所需的全部数据空间的大小及相互位置，那么，能在编译时安排好目标程序运行时全部数据对象在存储空间中的位置，且不随目标程序的运行而改变，则称这种分配为静态存储分配，如先于 FORTRAN 77 版本的 FORTRAN 语言采用这种分配策略。

如果一个程序语言允许递归过程、可变数组或动态数据结构，那么，就需采用动态存储分配策略。它有两种方式：栈式和堆式。栈式动态存储分配策略适用于组织 PASCAL, C, ALGOL 之类的语言的活动记录。每当调用一个过程产生一个活动时，它所需的活动记录就分配在栈顶。每当过程的一个活动结束时，就释放栈顶的活动记录。如果一个程序语言提供用户构造动态数据结构，有自由地申请数据空间和退还数据空间的机制（如 C++ 中的 new delete, PASCAL 的 new），或者不仅有过程而且有进程的程序结构，即空间的使用未必服从“先申请后释放，后申请先释放”的原则，那么栈式的动态存储分配方案就不适用了，这种情况下通常使用一种称为堆式的动态存储分配方案。下面仅就一个简单的栈式分配，说明过程调用和过程说明的翻译。

考虑一种简单的程序语言结构：没有分程序结构，过程定义不嵌套，但允许过程递归调用。图 6-9 显示运行时的内存空间安排和活动记录结构。

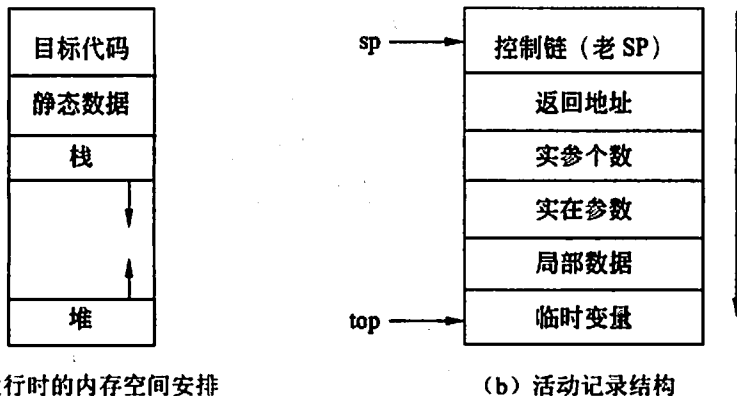


图 6-9 运行时的内存空间安排和活动记录结构

例如，对 C 语言，编译程序通常把全局量和每个函数中的静态量分在静态数据区，把一个函数中的自动变量和必需的数据组织成活动记录，随着调用函数的展开，分配在栈中。对于程序员构造动态数据结构，调用 malloc 动态申请的单元，则分配在堆中。执行目标代码时，sp 总是指向现行过程活动记录的起点，top 始终指向（已占用）栈顶单元。

正确答案:

(1) D (2) A (3) C (4) D (5) A

【例 6-18】在中间代码优化阶段,根据优化涉及到的程序范围,优化分为(1)优化、循环优化和全局优化。所谓(1)优化,是指在一个基本块内实行的优化,可以通过把一个基本块的四元式序列表示成(2)来实现。针对循环主要采用的优化措施有代码外提,删除归纳变量和(3)。

可选答案:

(1): A. 表达式 B. 过程内 C. 局部 D. 过程之间

(2): A. 有向图 B. 无向图 C. 有向无环图(DAG)

D. 有向完全图

(3): A. 复写传播 B. 变换计算顺序 C. 删除无用赋值 D. 强度削弱

分析:

优化是对程序进行等价(指不改变程序的运行结果)变换,经变换后的程序能生成更有效(运行时间更短、占用空间更小)的目标代码。原则上讲,优化可以在编译的各个阶段进行,但最主要的一类是对中间代码进行的优化,这类优化不依赖于具体的计算机。

根据优化所涉及的程序范围,分为局部优化、循环优化和全局优化三个不同的级别。

基本块上的优化可以按下面的方法实现:第一步,把基本块内的四元式序列转换成带标记或附加信息的无环路的有向图(简称DAG);第二步,根据第一步得到的DAG,按原来构造结点的顺序,重新写成新的四元式序列。这样,可以实现的优化有合并已知量,删除无用赋值和删除多余运算。

循环是一个很重要的可优化的地方,针对循环能采用的优化技术有代码外提、强度削弱和归纳变量删除。

代码外提是把循环体中不随循环的执行改变计算结果的表达式外提到循环的前置块中。

强度削弱是用较弱的运算代替较强的运算,例如用加法运算代替乘法运算。这项优化实施的对象是归纳变量,用加法实现归纳变量随循环的增量运算。为了实施这项优化技术,要先求出归纳变量,什么是归纳变量呢?若在循环体中变量 $i$ 只有惟一的形式 $i:=i+c$ ,其中 $c$ 是常量,则称 $i$ 是基本归纳变量。若在循环体中有另一个变量 $j$ ,其值是基本归纳变量的线性函数,则称 $j$ 是普通归纳变量。

删除归纳变量分两步进行:(1)对用于循环控制的基本归纳变量 $i$ ,用最简单依赖 $i$ 的归纳变量代替;(2)在循环体中进行复写传播,把无用归纳变量的赋值删去。

正确答案:

(1) C (2) C (3) D

## 6.3 思考练习题及答案

### 思考练习题

1. 最早体现结构化程序设计思想的程序设计语言是(1)，最早使用 BNF 文法定义程序设计语言语法的语言是(2)，最早提出类 (CLASS) 的概念的语言是(3)，用于函数式程序设计的程序设计语言是(4)，用于逻辑式程序设计的程序设计语言是(5)。

可选答案：

- (1) ~ (5): A. ADA                      B. PASCAL                      C. SMALLTALK  
    D. SNOBOL                      E. C                                  F. ALGOL 60  
    G. Java                              H. LISP                              I. PROLOG  
    J. FORTRAN                      K. SIMULA

2. 面向对象程序设计以(1)为基本的逻辑构件，用(2)来描述具有共同特征的一组对象，以(3)为共享机制，共享类中的方法和数据。

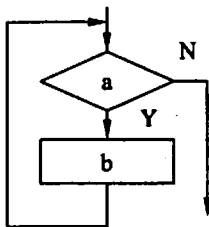
可选答案：

- (1) A. 模块                      B. 对象                              C. 结构                              D. 类  
       (2) A. 类型                      B. 抽象                              C. 类                                  D. 数组  
       (3) A. 引用                      B. 数据成员                      C. 成员函数                      D. 继承

3. 在下面的流程图中，如果标记为 b 的运算执行了 m 次 ( $m > 1$ )，那么标记为 a 的运算执行次数为(1)。

可选答案：

- (1): A. 1                              B. m-1                              C. m                                  D. m+1



4. 在下面的主程序中，实参 a 与形参 x 按引用调用 (call by reference) 的方式传递信息，实参 b 与形参 y 按值调用 (call by value) 的方式传递信息，那么，从过程 compute 返回主程序后 a 和 b 的值分别为(1)。

```
main program ;                      procedure compute (x,y);
```

```

{ a:=1;          { x:=x+2*y;
  b:=2;          y:=2*x+y;
  compute(a,b);    return;
}                }

```

可选答案:

(1): A. 5 和 2    B. 5 和 12    C. 1 和 2    D. 1 和 12

5. 消息传递是对象间通信的手段, 一个对象通过向另一个对象发送消息来请求其服务。一个消息通常包括 (1)。

可选答案:

(1): A. 发送消息的对象的标识、调用的发送方的操作名和必要的参数  
 B. 发送消息的类名和接受消息的类名  
 C. 接受消息的对象的标识、调用的接受方的操作名和必要的参数  
 D. 接受消息的类名

6. 在 C++ 语言中引进了类的概念。说明类的成员要说明其访问控制属性, 分为 (1), (2) 和 protected, 具有 (1) 访问控制属性的成员只能在本类中被访问; 具有 (2) 访问控制属性的成员在类的外部也可以访问。类具有 (3) 和 (4)。有了 (3) 可以隐藏类对象内部实现的复杂细节, 有效地保护内部所有数据不受外部破坏; (4) 增强了类的共享机制, 实现了软件的可重用性。

可选答案:

(1)、(2): A. operator    B. public    C. templete  
           D. virtual    E. friend    F. private  
 (3)、(4): A. 开放性    B. 封装性    C. 兼容性  
           D. 继承性    E. 多态性    F. 可扩充性

7. C++ 支持 (1), (1) 通过 (2) 实现。利用 (1) 和 (2) 的程序设计无需使用 switch 逻辑, 能设计实现易于扩展的软件系统。

可选答案:

(1)、(2): A. 模板类    B. 模板函数    C. 虚函数  
           D. 封装性    E. 继承性    F. 多态性

8. 高级语言的语言处理程序分为解释程序和编译程序两种。编译程序的工作从逻辑上一般由六个阶段组成, 而解释程序通常缺少 (1) 和 (2)。编译程序中设置 (2) 的主要目的是使最后生成的目标代码更有效。与编译系统相比, 解释系统 (3)。解释程序处理语言程序时, 大多数采用 (4) 方法。(5) 是一种典型的解释型语言。

可选答案:

(1)、(2): A. 词法分析    B. 语法分析    C. 语义分析  
           D. 中间代码生成    E. 代码优化    F. 目标代码生成



(3): A. 比较简单, 执行速度快    B. 比较简单, 执行速度慢

C. 比较复杂, 执行速度快    D. 比较复杂, 执行速度慢

(4): A. 源程序语句被逐个直接解释执行

B. 先将源程序转化成某种中间代码, 然后对这种表示解释执行

C. 先将源程序转化成目标代码, 再执行

D. 以上方法都不是

(5): A. BASIC

B. C

C. FORTRAN

D. PASCAL

9. 已知正规表达式  $r = (0|1)^*00$ , (1) 在  $L(r)$  中, 和  $r$  等价的非确定的有限自动机 NFA  $M$  是 (2), 和  $r$  等价的确定的有限自动机 DFA  $M$  是 (3)。

可选答案:

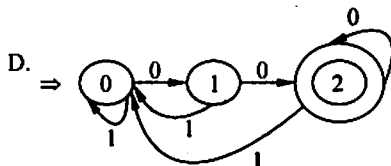
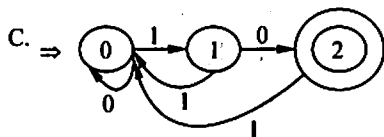
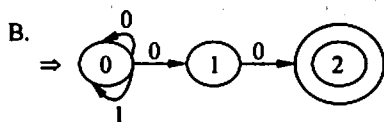
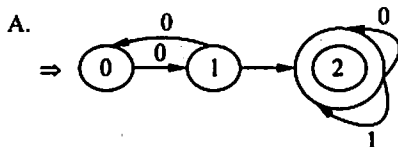
(1): A. 0000

B. 0001

C. 0010

D. 0011

(2)、(3):



10. 已知正规表达式  $r = (0|1)^*00$  的一个非确定的有限自动机 NFA  $M$  如图 6-10 所示, 其确定化过程由表 6-7 给出。在供选择的答案中选出合适的选项填到表 6-7 的默认项中。

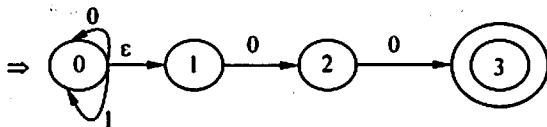


图 6-10 一个 NFA  $M$

表 6-7 图 6-10 的 NFA M 的确定化过程

	$I_0$	$I_1$
{0,1}	(1)	{0,1}
{0,1,2}	{0,1,2,3}	(2)
{0,1,2,3}	(3)	{0,1}

可选答案:

- (1) ~ (3): A. {0,1} B. {0,2} C. {0,3} D. {1,2} E. {1,3}  
 F. {2,3} G. {0,1,2} H. {0,1,3} I. {1,2,3} J. {0,1,2,3}

11. 巴科斯-诺尔范式 (即 BNF) 是一种广泛采用的 (1) 的工具。已知某文法  $G$  的规则集为  $\{A \rightarrow bA \mid cc\}$ , (2) 是  $L(G)$  的句子。(3) 是  $L_1 = \{a^n b^n \mid n \geq 1\}$  的文法的规则集, (4) 是  $L_2 = \{a^m b^n \mid m, n \geq 1\}$  的文法的规则集。(3) 是 (5) 文法。

可选答案:

- (1): A. 描述规则 B. 描述语言 C. 描述句子 D. 描述文法  
 (2): A. cc B. bc bc C. bbbcc D. bccbcc  
 (3)、(4): A.  $\{S \rightarrow CD, C \rightarrow bC \mid b, D \rightarrow aD \mid a\}$  B.  $\{S \rightarrow aSb \mid ab\}$   
 C.  $\{S \rightarrow CD, C \rightarrow aC \mid a, D \rightarrow bD \mid b\}$  D.  $\{S \rightarrow aSb \mid \varepsilon\}$   
 (5): A. 短语 B. 上下文有关 C. 上下文无关 D. 线性

12. 假设某程序语言的文法如下:

$$S \rightarrow a \mid b \mid (T)$$

$$T \rightarrow T d S \mid S$$

其中:  $V_T = \{a, b, d, (, )\}$ ;  $V_N = \{S, T\}$ ;  $S$  是开始符号。

考察该文法, 句型  $(S d (T) d b)$  是  $S$  的一个 (1)。其中 (2) 是句柄; (3) 是最左素短语; (4) 是该句型的直接短语, (5) 是短语。

可选答案:

- (1): A. 最左推导 B. 最右推导 C. 规范推导 D. 推导  
 (2): A.  $S$  B.  $b$  C.  $(T)$  D.  $S d (T)$   
 (3): A.  $S$  B.  $b$  C.  $(T)$  D.  $S d (T)$   
 (4): A.  $S$  B.  $S, (T), b$   
 C.  $S, (T) T d S, b$  D.  $(S d (T) d b)$   
 (5): A.  $(S d (T) d b)$  B.  $d (T)$   
 C.  $T d$  D.  $S d (T) d$

提示:

从  $S$  出发, 句型  $(S d (T) d b)$  不能完全由最左推导或最右推导推出:

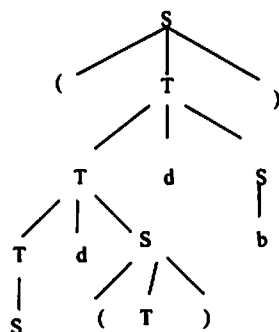
$S \Rightarrow (T)$

最右推导

$\Rightarrow (TdS)$             最右推导  
 $\Rightarrow (Tdb)$             最右推导  
 $\Rightarrow (TdSdb)$           最右推导  
 $\Rightarrow (Td(T)db)$         最右推导  
 $\Rightarrow (Sd(T)db)$         最左推导

或

$S \Rightarrow (T)$             最左推导  
 $\Rightarrow (TdS)$             最左推导  
 $\Rightarrow (TdSdS)$           最左推导  
 $\Rightarrow (SdSdS)$           最左推导  
 $\Rightarrow (Sd(T)dS)$         非最左推导  
 $\Rightarrow (Sd(T)db)$         最右推导



因此,句型  $(Sd(T)db)$  是  $S$  的一个推导。右边的分析树是回答本题其他几问的很好工具。

13. 在编译程序中,语法分析的方法有自底向上分析和自顶向下分析。自底向上分析方法自左向右扫描输入符号串,通过(1)方法分析其语法是否正确。例如,(2)就是一种自底向上的分析方法,与其他自底向上分析方法不同,它是根据(3)来进行归约的。自顶向下分析方法从文法的开始符号出发,判断其能否(4)出输入符号串。采用自顶向下分析方法时,要求文法不含有(5)。

可选答案:

- |                 |             |          |          |
|-----------------|-------------|----------|----------|
| (1): A. 归约—移进   | B. 移进—移进    |          |          |
| C. 移进—归约        | D. 归约—归约    |          |          |
| (2): A. 算符优先分析法 | B. 预测分析法    |          |          |
| C. 递归子程序分析法     | D. LL(1)分析法 |          |          |
| (3): A. 短语      | B. 素短语      | C. 直接短语  | D. 句柄    |
| (4): A. 归纳      | B. 归约       | C. 推理    | D. 推导    |
| (5): A. 右递归     | B. 左递归      | C. 直接右递归 | D. 直接左递归 |

14. 编译程序中语法分析器接受以(1)为单位的输入,并产生有关信息供以后各阶段使用。(2),(3)和 LR 分析法是几种常见的语法分析技术,其中(2)和(3)属于自顶向下分析方法,采用(3)主要是构造一张分析表;而 LR 分析法属于自底向上分析方法,LR 分析法主要有 SLR(1)、LR(0)、LR(1)、LALR(1)四种,其中(4)的分析能力最强,(5)的分析能力最弱。

可选答案:

- |                      |            |        |       |
|----------------------|------------|--------|-------|
| (1): A. 表达式          | B. 单词      | C. 产生式 | D. 语句 |
| (2)、(3): A. LL(1)分析法 | B. 算符优先分析法 |        |       |

## C. 弱优先分析法 D. 递归子程序法

(4)、(5): A. SLR (1) B. LR (0) C. LR (1) D. LALR (1)

15. 运行时的存储分配策略在设计语言的处理系统中是非常重要的。采用什么样的存储分配策略的依据是数据对象名字的(1)。如果在编译时就能确定目标程序运行时所需的全部数据空间的大小和相互位置,没有递归调用和建立动态数据结构,则可采用(2)。如果一个程序语言允许递归过程和可变数据结构,但名字的作用域满足块结构语言的规则,而过程的任何两个活动的生存期,要么是不重叠的,要么是嵌套的,则可采用(3)。如果一个程序语言为用户提供自由地申请数据空间和退还数据空间的机制,或者不仅有过程而且有进程的程结构,这种情况下通常使用(4)。

可选答案:

- (1): A. 类型 B. 结构  
C. 作用域和生存期 D. 之间的相互关系  
(2) ~ (4): A. 栈式分配 B. 最佳分配 C. 堆式分配  
D. 静态分配 E. 随机分配

16. 目前应用最广的语义分析方法是(1),其基本思想是把表示语言结构的每个文法符号都赋予若干个(2)以表示该文法符号的语义,而(2)值的计算以语义子程序的形式赋予相应的(3)。在语法分析的推导或归约的过程中,使用哪个(3),就调用哪个(3)的语义子程序完成(2)值的计算,即完成语义分析和翻译的任务。

可选答案:

- (1): A. 逐句翻译 B. 语法制导翻译  
C. 一对一翻译 D. 按意翻译  
(2): A. 类型 B. 地址 C. 属性 D. 中间代码  
(3): A. 文法符号 B. 非终结符号  
C. 终结符号 D. 产生式

17. 在编译程序中设置生成中间代码的主要目的是(1),常用的中间代码形式有后缀形式(也称逆波兰形式),三地址代码(包括三元式、四元式、间接三元式)和(2)。赋值语句  $a := b * (c + d) - e$  的后缀形式是(3)。

可选答案:

- (1): A. 便于编译程序的组织 B. 便于编译程序的移植  
C. 利于存储空间的组织 D. 利于目标代码的优化  
(2): A. 前缀形式 B. 中缀形式  
C. 树形表示 D. 图形表示  
(3): A.  $a := b * c + d - e$  B.  $abcd + * e - :=$   
C.  $a := bcd + * - e$  D.  $a := bcd + * e -$



18. 从优化涉及的范围的角度, 可将中间代码优化分为局部优化、循环优化和全局优化。考查一个 (1) 就可完成的优化称为局部优化, 可如下实现局部优化, 首先把 (1) 中的四元式构造造成 (2), 然后从 (2) 重写四元式。针对循环的优化是非常重要的, 采用的主要优化措施有循环不变计算的代码外提, 强度削弱和 (3)。

可选答案:

- |                    |                    |
|--------------------|--------------------|
| (1): A. 语句         | B. 程序段             |
| C. 函数              | D. 基本块             |
| (2): A. DAG        | B. Completed Graph |
| C. Connected Graph | D. Eulerian Graph  |
| (3): A. 删除公共子表达式   | B. 复写传播            |
| C. 删除无用赋值          | D. 删除归纳变量          |

### 思考练习题答案

1. (1) B (2) F (3) K (4) H (5) I
2. (1) B (2) C (3) D
3. (1) D
4. (1) A
5. (1) C
6. (1) F (2) B (3) B (4) D
7. (1) F (2) C
8. (1) D (2) E (3) B (4) B (5) A
9. (1) A (2) B (3) D
10. (1) G (2) A (3) J
11. (1) D (2) C (3) B (4) C (5) C
12. (1) D (2) A (3) C (4) B (5) A
13. (1) C (2) A (3) B (4) D (5) B
14. (1) B (2) D (3) A (4) C (5) B
15. (1) C (2) D (3) A (4) C
16. (1) B (2) C (3) D
17. (1) D (2) C (3) B
18. (1) D (2) A (3) D

## 第7章 系统开发与运行

### 7.1 内容提要

#### 7.1.1 软件工程概述

##### 1. 内容要点

- (1) 软件由计算机程序、数据及文档组成。
- (2) 软件与硬件、数据库、人、过程等共同构成计算机系统。
- (3) 软件工程定义为“建立并使用完善的工程化原则，以较经济的手段获得能在实际机器上有效运行的可靠软件的一系列方法。”
- (4) 软件工程包括三个要素：方法、工具和过程。
- (5) 软件生存期包含三个阶段：一是软件定义阶段，包括系统分析、软件计划，需求分析和定义；二是软件开发阶段，包括软件设计，程序编码，软件测试；三是软件运行和维护阶段。
- (6) 常用的软件生存期模型有瀑布模型、演化模型、螺旋模型、喷泉模型等。

##### 2. 学习难点

- (1) 软件的发展经历了程序设计、程序系统和软件工程等三个阶段。软件工程概念的出现源自软件危机。软件工程吸取和借鉴了人们长期以来从事各种工程项目所积累的行之有效的原理、概念、技术和方法。
- (2) 软件生存期模型反映软件生存期内各种工作应如何组织，软件生存期周期各个阶段应如何衔接。它是软件工程思想的具体化，是跨越软件生存期的系统开发、运行、维护所实施的全部活动和任务的过程框架。
- (3) 在软件开发过程中必须遵循的软件工程原则有8个：抽象与自顶向下、逐层细化；信息隐蔽和数据封装；模块化；局部化；确定性；一致性和标准化；完备性和可验证性。
- (4) 软件工程的基本原理有7个：按软件生存期分阶段制定计划并认真实施；坚持进行阶段评审；坚持严格的产品控制；使用现代程序设计技术；明确责任，使得工作结果能够得到清楚的审查；用人少而精；不断改进开发过程。

#### 7.1.2 系统分析与软件项目计划

##### 1. 内容要点

- (1) 基于计算机的系统为“某些元素的一个集合或排列，这些元素被组织起来以实

现某种方法，过程或借助处理信息进行控制。”

(2) 基于计算机系统的系统元素有硬件、软件、人、数据库、文档和过程（或环境）等。

(3) 计算机系统工程是一个问题求解活动，目的是揭示、分析所期望的功能，并把它们分配到各个单独的系统元素中去。

(4) 系统分析目标：识别用户要求、评价系统可行性、进行经济和技术分析、把功能分配给硬件、软件、人、数据库和其他系统元素、建立成本和进度限制、生成系统规格说明。

(5) 系统分析确定软件的范围（分配给软件元素的功能和性能）和总体要求，软件的外部接口，包括硬件、支撑软件、相关软件，人的接口，确定各种约束和限制。

(6) 可行性研究的目的是探讨软件项目是否值得立项，是否能够立项。它包括技术可行性、经济可行性、使用可行性和法律可行性等。提出可供选择的方案。

(7) 项目开发计划包括待开发系统概述、人员组织与安排、进度计划、资源利用、经费使用等。

## 2. 学习难点

(1) 在基于计算机的系统中，软件已取代硬件，成为软件项目规划中最困难的，最不易成功（按时并不超出成本）的，管理最具风险的系统元素。

(2) 系统分析员必须通过识别所希望的功能和性能范围来“界定”系统。

(3) 一旦确定了系统的功能、性能、约束和接口，系统分析下一步的任务就是将功能“分配”给一个或多个系统元素（即软件、硬件、人等）。往往还会提出一些候选方案供评价。

(4) 硬件工程、软件工程、人类工程和数据库工程的作用就是细化功能和性能的范围，产生一个能与其他系统元素适当集成的可操作的系统元素。

(5) 系统分析是一项包括许多任务（它们总称为计算机系统工程）的活动。由于这个术语常常用来暗指仅为软件需求分析活动的环境，故经常会产生混淆。

(6) 可行性研究并不保证在经济上显著合算、技术风险很低、法律问题很少，且选择的方案也不一定是最合理的。

(7) 经济可行性研究涉及成本—效益分析、长期的公司经营策略、对其他单位或产品的影响、开发所需的成本和资源，以及潜在的市场前景。

(8) 技术可行性要考虑开发的风险、资源的有效性和具备的技术基础。

(9) 法律可行性所涉及的范围包括：合同、责任、侵犯，以及其他一些技术人员常常不了解的陷阱。

(10) 可行性研究并不能代替需求分析。

### 7.1.3 需求分析

#### 1. 内容要点

(1) 需求分析的任务是确定软件系统的功能、性能、接口等要求；分析软件系统的数据要求；导出系统的逻辑模型，修正项目开发计划。如有必要，开发一个原型系统。

(2) 需求分析的基本原则是：能够表达和理解问题的信息域和功能域；以层次化的方式对问题进行分解和不断细化；要给出系统的逻辑视图和物理视图。

(3) 分析过程是：首先分析现实世界，理解当前系统是如何运行的，用一个具体模型（物理模型）来反映自己对当前系统的理解。在理解当前系统“怎样做”的基础上，抽取其“做什么”的本质，从而从当前系统的物理模型抽象出当前系统的逻辑模型。分析目标系统与当前系统逻辑上的差别，明确目标系统到底要“做什么”，从而导出目标系统的逻辑模型。

(4) 结构化分析方法最初是着眼于数据流，考虑数据流在系统中的传递和变换，自顶向下，逐层分解，建立系统的处理流程，以数据流图和数据词典为主要工具，建立系统的逻辑模型。

(5) 扩充的结构化分析方法将建模技术扩展到数据建模、功能建模和行为建模，以实体—关系图、数据流图和控制流图、状态—迁移图工具，数据词典为核心，从不同视点建立系统的分析模型。

(6) 数据流图用来表达系统内数据的运动情况，结构化语言、判定表与判定树用来描述数据流的变换，数据词典用来定义与系统相关的数据元素。

(7) 结构化语言也叫做程序设计语言（Program Design Language），简称 PDL，是一种介于自然语言和形式化语言之间的半形式化语言。

(8) 对于不太复杂的判定条件，或者使用判定表有困难时，使用判定树较好。而在一个加工逻辑中，如同时存在顺序、判断和循环时，使用结构化语言较好。而对于复杂的判定，组合条件较多，则使用判定表较好。

(9) 需求分析阶段需要提交的文档有软件需求说明书、数据要求说明书、初步的用户手册、确认测试计划，修改和完善软件开发计划。

(10) 评审的主要内容有系统定义的目标是否与用户的要求一致；需求分析提供的文档资料是否齐全；文档中的所有描述是否完整、清晰、准确反映用户要求；与其他系统成分的重要接口是否都已经描述；被开发项目的数据流与数据结构是否足够、确定。

#### 2. 学习难点

(1) 软件需求分析工作也是一个不断认识和逐步细化的过程。该过程将软件计划阶段所确定的软件功能和性能逐步细化到可详细定义的程度，并找到可行的解决方法。

(2) 画数据流图的基本步骤概括地说，就是自外向内，自顶向下，逐层细化，完善



求精。

(3) 用数据流图来描述数据处理过程中的数据加工情况。对于稍为复杂的实际问题, 在数据流图上常常出现十几个甚至几十个加工, 可建立分层的数据流图。

(4) 数据流图的每个加工至少有一个输入数据流和一个输出数据流。

(5) 任何一个数据流子图必须与它上一层的一个加工对应, 两者的输入数据流和输出数据流必须一致。此即父图与子图的平衡。

(6) 加工规格说明用来说明 DFD 中数据加工的输入, 实现加工的算法以及产生的输出。另外, 加工规格说明指明加工的约束和限制, 与加工相关的性能要求, 以及影响加工的实现方式的设计约束。

(7) 控制规格说明仅描述了系统的行为, 不提供被激活的加工的内部工作的细节。

(8) 当某一层数据流图中的数据存储对象不是父图相应加工的数据接口, 而只是本层数据流图中某些加工的之间的数据接口, 这样的数据存储应是局部数据存储。只有当局部数据存储作为某些数据加工之间的数据接口或某个特定加工的输入/输出时, 才把它画出来, 这样做有利于实现信息隐蔽。

## 7.1.4 软件设计

### 1. 内容要点

(1) 软件设计从技术上分成体系结构设计、数据设计、接口设计、过程设计 4 方面的工作, 从管理角度分为概要设计、详细设计两个阶段。

(2) 软件设计的目标有三: ①必须实现分析模型中描述的所有显式需求, 必须满足用户希望的所有隐式需求。②必须是可读、可理解的, 使得将来易于编程、易于测试、易于维护。③应从实现角度出发, 给出与数据、功能、行为相关的软件全貌。

(3) 软件设计基本原则有抽象化和逐层细化、系统模块化、程序结构化、信息隐蔽等。

(4) 模块独立性的量度有两个: 模块间的耦合和模块的内聚。为使系统模块独立性强必须做到高内聚低耦合。

(5) 用结构化设计方法建立的系统结构有变换型和事务型等两种。

(6) 结构化程序设计强调使用具有单入口单出口的基本控制结构, 通过组合嵌套, 形成程序的控制结构。尽可能避免使用会使程序质量受到影响的 GOTO 语句。

(7) 在程序设计过程中尽量采用自顶向下和逐步细化的原则, 由粗到细, 一步步展开。

(8) 软件详细设计的图形描述工具包括程序流程图、盒图和 PAD。

(9) 面向数据结构的 Jackson 方法的主要步骤有建立输入/输出数据结构、找出输入/输出数据结构中有对应关系的单元、导出程序结构、列出并分配操作和条件、用伪码写出程序的过程性描述。

### 2. 学习难点

(1) 从技术角度来看, 软件体系结构设计根据数据流图得到, 数据设计根据实体—

关系图、数据对象描述和数据词典得到,接口设计源自于数据流图,而过程设计则根据状态—转换图、控制规格说明和加工规格说明得来。

(2) 数据设计的基本原则:软件系统化方法可用于数据设计、确定所有数据结构的抽象数据类型、建立数据词典、低层数据设计应推迟到设计过程后期进行、数据结构表示应实现信息隐蔽、数据结构应成为可复用的、软件设计和程序设计语言应支持抽象数据类型的定义和实现。

(3) 软件过程的三种图形工具的五种基本控制结构是顺序结构、IF 两分支选择结构、CASE 多分支选择结构、先判断重复结构(DO-WHILE)、后判断重复结构(DO-UNTIL)。

(4) 软件设计的表格工具是判定表。判定表描述程序的静态逻辑,因此不能独立用于描述程序逻辑。主要用于检查程序的逻辑。

(5) 软件设计的语言工具是PDL语言,它是一种伪码,用于描述程序的逻辑。它的语法分为内外两层。外语法用高级语言中的关键字对程序进行分割,以描述程序结构和数据结构;内语法用自然语言描述各种操作和条件。

(6) HIPO是一种设计工具,主要包含可视目录表和IPO图。可视目录表给出程序的层次关系,IPO图则为程序各部分提供具体的工作细节。

(7) 在整个设计过程中,对各个时期的设计结果都需要经过一系列的设计质量评审,以便及时发现和及时解决在软件设计中出现的问题,防止把问题遗留到开发的后期阶段,造成后患。设计评审的内容包括:可追溯性、接口、设计风险、实用性、技术清晰度、可维护性、质量、各种选择方案、设计限制、其他具体问题的评估。

## 7.1.5 编码

### 1. 内容要点

(1) 程序设计语言是用于书写计算机程序的语言,是实现性的软件语言。

(2) 程序设计语言的研究语法、语义和语用。

(3) 程序设计语言的基本成分包括数据成分、运算成分、控制成分和传输成分。

(4) 程序设计语言的分类:按语言级别可分为低级语言与高级语言;按应用范围可分为通用语言和专用语言;按对用户要求可分为过程式语言和非过程式语言。按语言所包含的成分可分为顺序语言、并发语言、分布式语言等。

(5) 为特定开发项目选择程序设计语言需考虑的因素:应用领域、算法和计算的复杂性、软件运行的环境、用户需求、数据结构的复杂性、开发人员的水平等。

(6) 软件的设计质量与程序设计语言的技术性能无关(面向对象设计例外)。但在将软件设计转化为程序代码时,转化的质量往往受语言性能的影响。因而也会影响到设计方法。

(7) 一个好的程序应具有模块化结构,系统应有较高的模块独立性。在把设计转换成程序时,程序设计语言的特性就影响到这些设计概念。

(8) 程序设计风格的 4 个方面包括源程序文档化、数据说明的方法、语句结构和输入 / 输出方法, 力图从编码原则角度探讨提高程序可读性, 改善程序质量的方法和途径。

## 2. 学习难点

(1) 程序设计语言的语法用来表示构成语言的各个记号之间的组合规则。

(2) 程序设计语言的语义用来表示按照各种表示方法所表示的各个记号的特定含义。

(3) 程序设计语言的语用表示构成语言的各个记号和使用者的关系。

(4) 从应用领域看, COBOL 语言适合于商业领域的应用; FORTRAN 语言适合于科学与工程计算; PROLOG 和 LISP 适用于人工智能的应用; Smalltalk、C++ 和 Java 是面向对象的语言; C 开发系统的程序设计语言。

(5) Java 语言的特点有简单性、面向对象、分布性、强类型性、安全性、解释性、平台无关性、可移植性、多线程、动态性, 特别适合于网络编程。

(6) 第四代语言兼有过程性和非过程性的两重特性。程序员规定条件和相应的动作, 这是过程性的部分, 并且指出想要的结果, 这是非过程部分。然后由 4GL 语言系统运用它的专门领域的知识来填充过程细节。

(7) 从软件工程的角度来看, 汇编语言只是在高级语言无法满足设计要求时, 或者不具备支持某种特定功能 (例如特殊的输入 / 输出) 的技术性能时, 才被使用。

(8) 在编写程序时需注意源程序的文档化, 包括符号名的命名、加注解及程序的视觉组织等。

## 7.1.6 软件测试

### 1. 内容要点

(1) 软件测试是为了发现错误而执行程序的过程。

(2) 测试用例由测试输入数据和与之对应的预期输出结果组成。在设计测试用例时, 应当包括合理的输入条件和不合理的输入条件。

(3) 测试过程按 3 个步骤进行, 即单元测试、集成测试和确认测试。

(4) 白盒测试与黑盒测试的区分在于白盒测试是根据程序的内部逻辑设计测试用例, 黑盒测试是根据软件的各种规约设计测试用例。

(5) 逻辑覆盖是白盒测试的测试用例设计方法。它根据覆盖程度的不同, 可分为语句覆盖、判定覆盖、判定-条件覆盖、条件组合覆盖及路径覆盖。

(6) 黑盒测试的测试用例设计方法有等价类划分、边值分析、错误猜测、因果图等。

(7) 使用各种测试方法的综合策略是: ①在任何情况下都必须使用边值分析方法。用这种方法设计出测试用例发现程序错误的能力最强。②必要时用等价类划分方法补充一些测试用例。③用错误推测法再追加一些测试用例。④对照程序逻辑, 检查已有测试用例的逻辑覆盖程度。如果未达到要求的覆盖标准, 应再补充足够的测试用例。⑤如果程



序的功能说明中含有输入条件的组合情况,则一开始就可选用因果图法。

## 2. 学习难点

(1) 软件测试在软件生存期中横跨两个阶段:通常,编码与单元测试属于软件生存期中的同一个阶段。对软件系统进行各种综合测试则是测试阶段的工作。

(2) 软件开发过程是一个自顶向下,逐步细化的过程,而测试过程则是依相反的顺序安排的自底向上,逐步集成的过程。低一级测试为上一级测试准备条件。

(3) 单元测试需要依据详细设计说明书和源程序清单,了解该模块的 I/O 条件和模块的逻辑结构,主要采用白盒测试的测试用例,辅之以黑盒测试的测试用例。

(4) 集成测试是对由各模块组装而成的系统进行测试,检查各模块间的接口和通信。该测试主要发现设计中的问题,通常采用黑盒测试。它包括渐增式集成和非渐增式集成。

(5) 确认测试检查软件的功能、性能及其他特征是否与用户的要求一致。它以软件的需求说明书(亦称需求规约)为依据,通常采用黑盒测试。

(6) 在逻辑覆盖的各种覆盖准则中,以语句覆盖的覆盖程度最弱,路径覆盖的覆盖程度最强。

(7) 等价类划分是以各输入等价类的代表值作为测试数据;边值分析是以各输入/输出等价类的边界值或接近边界的值作为测试数据;因果图是考虑各等价类间的相互作用来设计测试数据。

## 7.1.7 面向对象方法

### 1. 内容要点

(1) “面向对象 = 对象 + 类 + 继承 + 消息通信”。如果一个软件系统是使用这样 4 个概念设计和实现的,则认为这个软件系统是面向对象的。

(2) 对象可以定义为系统中用来描述客观事物的一个实体,它是构成系统的一个基本单位。它是一组属性以及这组属性上的专用操作的封装体。

(3) 把具有相同特征和行为的对象归在一起就形成了类。类成为某些对象的模板,抽象地描述了属于该类的全部对象的属性和操作。

(4) 将几个类之间具有共性的东西(信息结构和行为)抽取出来放在一般类中,而将各个类的特有的东西放在特殊类中分别描述,则可建立起特殊类对一般类的继承。

(5) 多态性是指同一个操作作用于不同的对象上可以有不同的解释,并产生不同的执行结果。与它相关的一个概念就是动态绑定。

(6) Coad 和 Yourdon OOA 模型的建立过程有五个层次:主题层、对象&类层、结构层、属性层和服务层。相应五个活动:定义主题,标识对象&类,定义结构(包括分类结构和组装结构),定义属性及其实例连接,定义服务及其消息连接。

(7) Coad 和 Yourdon 的 OOD 模型有 4 个部分:问题领域部分、人机交互部分、任务管理部分、数据管理部分。相应有 4 个活动:设计问题领域部件,设计人机交互部件,



设计任务管理部件，设计数据管理部件。

(8) Booch 方法包含“微开发过程”和“宏开发过程”。微开发过程定义了一组任务，并在宏开发过程的每一步骤中反复使用它们，以维持演进途径。

(9) Booch 的 OOA 宏开发过程由 4 步构成：标识类和对象，确定类和对象的含义，标识它们之间的关系，说明每个类和对象的界面和实现。Booch 的 OOD 微开发过程由 3 步构成：规划系统的体系结构，确立设计方针和部署规划。

(10) Rumbaugh 等人提出的对象建模技术 (OMT) 用于分析、系统设计和对象设计。分析活动建立三个模型：对象模型（描述对象、类、层次和关系），动态模型（描述对象和系统的行为），功能模型（类似于高层的 DFD，描述穿越系统的信息流）。

## 2. 学习难点

(1) 对象的类型可以是以下 7 类：外部实体、信息结构、需要记忆的事件、角色、组织机构、位置、操作规程等。

(2) 类常常可看做是一个抽象数据类型 (ADT) 的实现。但更重要的是把类看做是表示某种概念的一个模型。事实上，类是单个的语义单元。

(3) 如果一个子类只用到一个父类的特征，这种继承为单一继承；如果一个子类可以从多个父类中继承，称为多重继承。

(4) 消息是一个对象与另一个对象的通信单元，是要求某个对象执行类中定义的某个操作的规格说明。通常消息有以下 4 类：发送对象激活接收对象；发送对象传送信息给接收对象；发送对象询问接收对象；发送对象请求接收对象提供服务。

(5) 动态绑定把函数调用与目标代码块的连接延迟到运行时进行。这样，只有发送消息时才与接收消息实例的一个操作绑定。

(6) 面向对象技术的特征是：方法的惟一性；从生存期的一个阶段到下一个阶段的高度连续性；把 OOA、OOD 和 OOP 集成到生存期的相应阶段。

(7) 建立 OOA 模型的 5 个基本原则：建立信息域模型；描述功能；表达行为；划分功能、数据、行为模型，揭示更多的细节；用早期的模型描述问题的实质，用后期的模型给出实现的细节。

(8) 高层设计的 3 条规则：最小化各部件间的通信；隐藏复杂性；逻辑功能分组。

(9) 类设计的方针包括：信息隐蔽；消息限制；狭窄界面；强内聚；弱耦合；显式信息传递；派生类当做派生类型；抽象类。

## 7.1.8 软件维护

### 1. 内容要点

(1) 软件维护的类型可分为改正性维护、适应性维护、完善性维护和预防性维护。

(2) 与软件维护有关的问题有 5 个方面：理解需要维护的软件是很困难的；需要维护的软件往往缺少合适的文档；维护软件时通常不能指望得到原来开发该软件人员的帮

助：多数软件在设计时没有考虑将来的维护；软件维护通常不是一个吸引人的工作。

(3) 修改程序的副作用有三种：修改代码的副作用、修改数据的副作用、文档的副作用。

(4) 软件可维护性是指软件能够被理解、改正、适应及增强功能的容易程度。可理解性、可测试性、可修改性是衡量软件可维护性的几个主要质量因素。

(5) 软件再工程，也叫做软件复壮（修理）或再生，是一类软件工程活动，它能够使我们：增进对软件的理解；准备或直接提高软件的可维护性、复用性或演化性。

(6) 软件的逆向工程是分析程序，力图在比源代码更高抽象层次上建立程序表示的过程。通常，软件逆向工程是设计恢复的过程。逆向工程工具可以从已存在程序中抽取数据结构、体系结构和程序设计信息。

## 2. 学习难点

(1) 在整个软件维护活动中，改正性维护约占 20%，完善性维护约占 50%，适应性维护约占 25%。

(2) 在整个软件生存周期所花费的代价中，1980 年代末用于维护的代价约为 75%，到 1990 年代初约为 90%。

(3) 提高可维护性需从五个方面着手：建立明确的软件质量目标和优先级、使用提高软件质量的技术和工具（如模块化方法、结构化技术、面向对象技术等）、进行明确的质量保证审查（包括在检查点进行复审、验收检查、周期性地维护审查、对软件包进行检查）、选择可维护的程序设计语言、改进程序的文档。

## 7.1.9 软件管理

### 1. 内容要点

(1) 软件项目管理包括进度管理、成本管理、质量管理、人员管理、资源管理、标准化管理。管理的对象是进度、系统规模及工作量估算、经费、组织机构和人员、风险、质量、作业和环境配置等。

(2) 完成软件项目所需的资源包括用以支持软件开发的工具（硬件及软件工具）和人。

(3) 软件成本估算是把待开发的软件细分，直到每一个子任务都已经明确所需要的开发工作量，然后把它们加起来，得到软件开发的总工作量，从而算出总成本。

(4) 软件开发项目成本估算的 Putnam 模型是一种动态多变量模型。结构型成本估算的 COCOMO 模型是一种静态单变量模型

(5) 对于较大的项目，需要采用图示的方法描述项目的进度，特别是表现各项任务之间进度的相互依赖关系。几种有效的图示方法包括甘特图和 PERT 图。

(6) 风险分析的主要活动有风险识别、风险估算、风险评价和风险管理。

(7) 风险分为项目风险、技术风险和商业风险。风险构成有性能风险、成本风险、

支持风险、进度风险。

(8) 程序设计小组的组织形式有三种：主程序员制小组、民主制（无主程序员）小组、层次式程序员小组。

## 2. 学习难点

(1) 在软件计划和需求分析阶段，主要工作是由管理人员和高级技术人员在做，初级技术人员参与较少。待到对软件进行具体设计、编码及测试时，管理人员逐渐减少对开发工作的参与，高级技术人员主要在设计方面把关，大量的工作将由初级技术人员去做。到了软件开发的后期，管理人员和高级技术人员又将投入很多的精力。

(2) 在 COCOMO 模型中，考虑开发环境，软件开发项目的总体类型可分为三种：组织型、嵌入型和介于上述两种软件之间的半独立型。

(3) 一个软件任务由一个人单独开发，生产率最高；而对于一个稍大型的软件项目，一个人单独开发，时间太长。因此软件开发组是必要的。但是软件开发组的规模太大，人数太多，会因人与人之间的通信造成生产率降低。

(4) 在软件工程项目中必须处理好进度与质量之间的关系。在进度压力下赶任务，其成果往往是以牺牲产品的质量作为代价的。

(5) 项目风险是指潜在的预算、进度、个人（包括人员和组织）、资源、用户和需求方面的问题，以及它们对软件项目的影响。技术风险是指潜在的设计、实现、接口、检验和维护方面的问题。5 种主要的商业风险是：建立的软件虽然很优秀但不是市场真正所想要的；建立的软件不再符合公司的整个软件产品战略；建立了销售部门不清楚如何推销的软件；由于重点转移或人员变动而失去上级管理部门的支持；没有得到预算或人员的保证。

(6) 建立项目组织的原则：尽早落实责任、减少接口、责权均衡。

## 7.1.10 软件质量保证

### 1. 内容要点

(1) ANSI/IEEE Std 729—1983 定义软件质量为“与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体”。

(2) 软件质量特性反映了软件的本质。讨论一个软件的质量，问题最终要归结到定义软件的质量特性。而定义一个软件的质量，就等价于为该软件定义一系列质量特性。

(3) McCall 质量模型中的质量概念基于 11 个特性之上，而这 11 个特性分别面向软件产品的运行、修正、转移。McCall 等认为，特性是软件质量的反映，软件属性可用做评价准则，定量化地度量软件属性可知软件质量的优劣。

(4) ISO/IEC 9126—1991 质量特性标准中，建立三层次的质量模型。第一层称为质量特性，第二层称为质量子特性，第三层称为度量。该标准定义了 6 个质量特性，并推荐了 21 个子特性，但不作为标准。



(5) 软件的质量保证就是为了向用户及社会提供满意的高质量的产品而进行的有计划、有系统的管理活动。它是面向消费者的活动。

(6) 软件质量保证由各项任务构成，这些任务的参与者有两种人：软件开发者和质量保证人员。前者负责技术工作，后者负责质量保证的计划、监督、记录、分析及报告工作。

(7) 软件开发人员通过采用可靠的技术方法和措施，进行正式的技术评审，执行计划周密的软件测试来保证软件产品的质量。软件质量保证人员则辅助软件开发组得到高质量的最终产品。这方面的工作包括：为项目制定 SQA 计划；参与开发该软件项目的软件过程描述；评审各项软件工程活动；审核指定的软件工作产品；记录软件工作及软件工作产品的偏差；

跟踪问题的解决以及协调变更的控制与管理，并帮助收集和分析软件度量的信息。

## 2. 学习难点

(1) 软件质量反映了三方面的问题：软件需求是度量软件质量的基础。不符合需求的软件就不具备质量。规定了一组开发准则用来指导软件开发，如果不遵守这些开发准则，软件质量就得不到保证。往往会有一些隐含的需求没有显式地提出来，如果软件只满足那些精确定义了的需求而没有满足这些隐含的需求，软件质量也不能保证。

(2) 软件质量模型的特点是：把软件质量特性定义成分层模型。最基本的叫做质量特性，它由一些质量子特性定义和度量。质量子特性又可由它的一些子特性定义和度量。

(3) 软件的质量保证活动，是涉及各个部门的部门间的活动。为了顺利开展质量保证活动，事先明确部门间的质量保证业务，建立部门间的联合与协作的机制十分重要，这个机制就是质量保证体系。

(4) 协调软件开发使得混乱减到最小的技术叫做配置管理。配置管理是一种标识、组织和控制修改的技术，目的是使错误达到最小并最有效地提高生产率。

(5) 软件维护和软件配置管理之间的区别是：维护是一组软件工程活动，它们发生于软件已交付给用户并已投入运行之后；软件配置管理是一组追踪和控制活动，它们开始于软件开发项目开始之时，结束于软件被淘汰之时。

(6) 任何一门工程技术都离不开标准，软件工程也不例外。软件产品标准定义了软件产品（或构件产品）应当具有的属性；软件过程标准定义了软件过程应如何进行；评审规程则规定了设计评审应怎样组织。

(7) 软件工程标准的类型有：过程标准、产品标准、专业标准和记法标准等。根据软件工程标准制定的机构和标准适用的范围有所不同，它可分为五个级别，即国际标准、国家标准、行业标准、企业（机构）标准及项目（课题）标准。

(8) 按照文档产生和使用的范围，软件文档大致可分为三类：开发文档、管理文档、用户文档。这些文档可归于以下 13 种：可行性研究报告、项目开发计划、软件需求说明书、数据要求说明书、概要设计说明书、详细设计说明书、用户手册、操作手册、测试



计划、测试分析报告、开发进度月报、项目开发总结报告、维护修改建议（问题报告）等。

### 7.1.11 软件开发工具与环境

#### 1. 内容要点

(1) 用来辅助软件开发、运行、维护、管理、支持等过程中的活动的软件称为软件工具。

(2) 按软件过程可将软件工具分为4类：支持软件开发过程的工具、支持软件维护过程的工具、支持软件管理过程和支持过程的工具、应用类工具。

(3) 软件开发工具是指支持软件产品开发的软件系统。它由软件工具集和环境集成机制构成。工具集包括支持软件开发相关过程、活动、任务的软件工具；环境集成机制为工具集成和软件开发、维护及管理提供统一的支持。

(4) 环境集成机制包括数据集成、控制集成和界面集成。

(5) 集成型开发环境是一种把支持多种软件开发方法和开发模型的软件工具集成在一起的软件开发环境，它通常由工具集和环境集成机制两部分组成。

(6) 集成型软件开发环境中除数据集成、控制集成和界面集成外，还可以有方法集成、过程集成、平台集成、工具集成等。

#### 2. 学习难点

(1) 通过环境集成机制，各工具用统一的数据接口规范存储或访问环境信息库，各工具采取统一的界面形式，保证各工具界面的一致性，同时为各工具或开发活动之间的通信、切换、调度和协同工作提供支持。

(2) 环境集成机制的核心是环境信息库。

(3) 数据集成提供统一的数据模式和数据接口规范，需要相互协同的工具通过这种统一的规范交换数据。数据集成可有共享文件、共享数据结构或共享信息库等不同的层次。

(4) 控制集成支持各工具或各开发活动之间的通信、切换、调度和协同工作，并支持软件开发过程的描述、执行和转接。通常使用消息传送的方式实现控制的集成。

(5) 界面集成为统一的工具界面风格、统一的操作方式提供支持，使得环境中的工具具有相同的视觉效果和操作规则。

## 7.2 例题分析

【例 7-1】软件开发模型用以指导软件的开发。演化模型是在快速开发一个 A 的基础上，逐步演化成最终的系统。螺旋模型综合了 B 的优点，并增加了 C。喷泉模型描述的是面向 D 的开发过程，反映了该开发过程的 E 特征。

可选答案:

- A: ① 模块                      ② 运行平台                      ③ 原型                      ④ 主程序  
B: ① 瀑布模型和演化模型                      ② 瀑布模型和喷泉模型  
     ③ 演化模型和喷泉模型                      ④ 原型和喷泉模型  
C: ① 质量评价                      ② 进度控制                      ③ 版本控制                      ④ 风险分析  
D: ① 数据流                      ② 数据结构                      ③ 对象                      ④ 构件  
E: ① 迭代和有间隙                      ② 迭代和无间隙  
     ③ 无迭代和有间隙                      ④ 无迭代和无间隙

正确答案: A: ③ B: ① C: ④ D: ③ E: ②

分析: 软件开发模型是指软件开发的全部过程、活动和任务的结构框架。主要的开发模型有瀑布模型、演化模型、螺旋模型、喷泉模型和智能模型。其中, 演化模型是在快速开发一个原型的基础上, 逐步演化成最终的系统。螺旋模型综合了瀑布模型和演化模型的优点, 并增加了风险分析。沿着螺旋线自内向外, 每旋转一圈, 就得到 B 的一个新版本。喷泉模型描述的是面向对象的开发过程, 反映了该开发过程的迭代和无缝隙特征。

【例 7-2】 软件需求分析的任务不应包括 A。进行软件需求分析可以使用多种工具, B 但是不适用的。在软件需求分析阶段中, 分析员要从用户那里解决的最重要的问题是 C。需求规格说明书的内容不应当包括 D。该文档在软件开发中具有重要的作用, 但其作用不应当包括 E。

可选答案:

- A: ① 问题分析    ② 信息域分析    ③ 结构化程序设计    ④ 确定逻辑模型  
B: ① 数据流图    ② 判定表                      ③ PAD 图                      ④ 数据词典  
C: ① 要让软件做什么                      ② 要给该软件提供哪些信息  
     ③ 要求软件工作效率如何                      ④ 要让软件具有什么样的结构  
D: ① 对重要功能的描述                      ② 对算法的详细过程性描述  
     ③ 软件确认准则                      ④ 软件的性能  
E: ① 软件设计的依据                      ② 用户和开发人员对软件要“做什么”的共同理解  
     ③ 软件验收的依据                      ④ 软件可行性分析的依据

正确答案: A: ③ B: ③ C: ① D: ② E: ④

分析: 软件需求分析的任务是通过与用户的合作, 了解用户对待开发系统的要求; 根据对用户要求的系统所在的信息域的调查、分析, 确定系统的逻辑模型; 并对求解的问题做适当的分解, 使之适合于计算机求解。需求分析的结果是软件需求规格说明书。

结构化程序设计是在详细设计和编码阶段所采用的技术, 而不是需求分析阶段要采用的技术。在需求分析阶段, 分析人员可以用数据流图描述系统的数据流的变换和流向, 用数据词典定义在数据流图中出现的数据流、数据文件、加工或处理, 用判定表表示复

杂条件和动作组合的情况。但 PAD 图是在详细设计阶段使用的描述加工逻辑的工具，不适用于需求分析。此外，软件需求分析阶段只确定软件系统要“做什么”，完成对重要功能、性能、确认准则的描述，至于“怎么做”由后续的设计阶段完成，对算法的详细过程性描述也是在设计阶段给出。软件可行性分析应在需求分析之前，所以需求分析规格说明不能成为可行性分析的依据。

【例 7-3】结构化分析方法是一种面向 A 的软件需求分析方法，该方法最常用的图形工具是 B，与其配合使用的是 C。B 中带有名字及方向的一种图形元素是 D，不能由计算机处理的成分是 E。

可选答案：

- A: ① 对象                      ② 数据结构                      ③ 数据流                      ④ 控制流  
B~C: ① 程序流程图              ② 实体关系图                      ③ 数据流图                      ④ 网络图  
          ⑤ 结构图                      ⑥ 数据字典  
D~E: ① 控制流                      ② 信息流                      ③ 数据流                      ④ 信号流  
          ⑤ 数据源/终点              ⑥ 结点

正确答案：A: ③    B: ③    C: ⑥    D: ③    E: ⑤

分析：结构化分析方法（SA）是一种面向数据流的软件分析方法，适用于开发数据处理型软件的需求分析。结构化分析方法使用的工具主要有数据流图（DFD）、数据字典（DD）、结构化语言、判定表和判定树。其中，数据流图以图形的方式表达数据处理系统中信息的变换和传递过程。与数据流图配合使用的是数据字典，它对数据流图中出现的所有图形元素给出逻辑定义。有了数据字典，使得数据流图中的数据流、加工和文件得到确切的解释。

通常在数据流图中，可能出现四种基本符号：数据流、加工、文件、数据源点和终点。数据流是具有名字和流向的数据，加工是对数据流的变换，文件是可访问的存储起来的数据，数据源点和终点是数据处理过程的数据来源和数据去向，它是不能用计算机处理的成分。

【例 7-4】在软件的开发过程中常用图作为描述工具。如 DFD 就是面向 A 分析方法的描述工具。在一套分层 DFD 中，如果某一张图中有 N 个加工（process），则这张图允许有 B 张子图。在一张 DFD 图中，任意两个加工之间 C。在画分层 DFD 时，应注意保持 D 之间的平衡。DFD 中从系统的输入流到系统的输出流的一连串连续变换形成一种信息流，这种信息流可分为 E。

可选答案：

- A: ① 数据结构    ② 数据流                      ③ 对象                      ④ 构件（component）  
B: ① 0                      ② 1                      ③ 1~N                      ④ 0~N  
C: ① 有且仅有一条数据流                      ② 至少有一条数据流  
          ③ 可以有 0 或多条名字互不相同的数据流



④ 可以有 0 或多条数据流, 但允许其中有若干条名字相同的数据流

- D: ① 父图与其子图                      ② 同一父图的所有子图  
      ③ 不同父图的所有子图              ④ 同一子图的所有直接父图  
 E: ① 控制流和变换流                    ② 变换流和事务流  
      ③ 事务流和事件流                    ④ 事件流和控制流

正确答案: A: ②    B: ④    C: ③    D: ①    E: ②。

分析: 在软件的开发过程中常用图作为描述工具。例如, 数据流图 (DFD) 就是面向数据流的分析方法的描述工具。因为在分层的 DFD 中, 如果某一加工有子图, 那么子图只有一个, 它是该加工的细化, 因此, 某一张图中有  $N$  个加工, 则这张图最多允许有  $N$  张子图。在一张 DFD 图中, 任意两个加工之间可以有 0 或多条名字互不相同的数据流。在画分层 DFD 时, 应注意保持父图和子图之间的平衡, 即父图与子图的输入/输出数据流必须保持一致, 以避免遗漏和错误。DFD 中从系统的输入流到系统的输出流的一连串连续变换形成一种信息流, 这种信息流可分为变换流和事务流。

【例 7-5】软件设计中划分模块的一个准则是 A。两个模块之间的耦合方式中, B 耦合的耦合度最高, C 耦合的耦合度最低。一个模块内部的内聚种类中 D 内聚的内聚度最高, E 内聚的内聚度最低。

可选答案:

- A: ① 低内聚低耦合    ② 低内聚高耦合    ③ 高内聚低耦合    ④ 高内聚高耦合  
 B: ① 数据              ② 非直接              ③ 控制              ④ 内容  
 C: ① 数据              ② 非直接              ③ 控制              ④ 内容  
 D: ① 偶然              ② 逻辑                  ③ 功能              ④ 过程  
 E: ① 偶然              ② 逻辑                  ③ 功能              ④ 过程

正确答案: A: ③    B: ④    C: ②    D: ③    E: ①。

分析: 在软件设计中划分程序模块的原则是尽可能提高模块的独立性, 就是尽可能做到高内聚和低耦合。在两个模块之间的耦合方式中, 内容耦合的耦合程度最高, 非直接耦合的耦合程度最低, 数据耦合的耦合程度也比较低。在模块的所有内聚种类中, 功能内聚的内聚度最高, 偶然内聚的内聚度最低。

【例 7-6】软件设计的常用方法有 SD 方法、Jackson 方法、Parnas 方法等。Jackson 方法是一种面向数据结构的设计方法, 一般在数据处理中, 数据结构有 A、B、C 三类, 并根据 D 来导出程序结构。Parnas 方法的主要思想是 E, 这是提高可维护性的重要措施。

可选答案:

- A~C: ① 表              ② 顺序              ③ 集合              ④ 图                  ⑤ 选择  
       ⑥ 重复              ⑦ 树                  ⑧ 栈                  ⑨ 队列              ⑩ 堆  
 D: ① 数据结构    ② 数据间的控制结构    ③ 数据流图    ④ IPO 图



E: ① 结构化      ② 模块化      ③ 信息隐蔽      ④ 抽象化

正确答案: A: ②    B: ⑤    C: ⑥    D: ①    E: ③。其中, A、B、C 的答案顺序可互换。

分析: 面向数据结构的软件设计方法认为数据结构不仅影响软件的结构, 而且左右软件的过程层面, 如重复性数据总是用循环控制机制来处理, 任选性数据总是用判断控制机制来处理。因而面向数据结构的软件设计方法可根据数据结构表示直接导出软件的过程性表示。Jackson 方法就是一种面向数据结构的設計方法。它主张程序结构应与问题结构相对应, 根据问题的数据结构来导出程序结构。Jackson 指出, 问题应分解成可用三种结构形式表示的各成分的层次结构。三种结构形式是顺序、选择和重复。

Parnas 方法主张在分解模块时就应采取措旆使将来因修改造成的影响尽可能局限在一个或少数几个模块的内部, 从而提高软件的可维护性。为达到这一目的, Parnas 提出了信息隐蔽原则。根据该原则, 概要设计的步骤如下: (1) 列出将来可能发生变化的因素; (2) 划分模块时将一些可能发生变化的因素隐含在某个模块的内部, 使其他模块与这些因素无关。就是说, 将这些因素隔离于某个模块内部, 这些因素的变化不传播到所在模块的边界之外。

【例 7-7】 Jackson 结构化程序设计方法是英国的 M. Jackson 提出的, 它是一种面向 A 的程序设计方法, 主要适用于规模适中的 B 系统的开发, 其程序开发的基本步骤依次是 C、D、E。

可选答案:

A: ① 对象      ② 数据流      ③ 数据结构      ④ 控制结构

B: ① 数据处理      ② 文字处理      ③ 实时控制      ④ 科学计算

C~E: ① 建立数据结构      ② 列出基本操作

③ 建立程序结构      ④ 建立控制结构      ⑤ 建立对象

正确答案: A: ③    B: ①    C: ①    D: ③    E: ②

分析: Jackson 结构化程序设计方法 (简称 JSP) 是一种面向数据结构的程序设计方法。它主张程序结构应与问题结构相对应。对一般数据处理系统而言, 问题结构可用它所处理的数据结构来表示, 因而要求程序结构反映出数据结构。JSP 方法的基本步骤是: (1) 考察问题的环境, 分析所要处理的数据, 做出数据结构图; (2) 以数据结构为基础导出程序结构; (3) 列出进行处理所需要的基本操作, 并把这些操作分配给程序结构中的适当部分。

JSP 主要适用于规模适中的数据处埋类问题, 尤其是企事业管理系统。

【例 7-8】 结构化设计方法在软件开发中用于 A, 它是一种面向 B 的设计方法。该方法使用的图形工具是 C, C 中的矩形表示 D。如果两个矩形之间有直线相连, 表示它们存在 E 关系。

可选答案:

- A: ① 测试用例设计    ② 概要设计    ③ 程序设计    ④ 详细设计  
B: ① 对象    ② 数据结构    ③ 数据流    ④ 控制流  
C: ① 系统结构图    ② 数据流程图    ③ 程序流程图    ④ 实体关系图  
D: ① 数据    ② 加工    ③ 模块    ④ 存储  
E: ① 链接    ② 调用    ③ 并列    ④ 顺序执行

正确答案: A: ②    B: ③    C: ①    D: ③    E: ②

分析: 结构化设计方法(SD)是由 Yourdon 和 Constanting 提出的,它是结构化分析方法(SA)的后续阶段,用于软件的概要设计。SD 是一种面向数据流的设计方法,适用于变换型或事务型数据处理系统。在设计过程中,它从整体的程序结构出发,突出了程序的模块化,利用系统结构图表达程序模块之间的关系。

系统结构图是采用结构化设计方法进行软件概要设计的重要描述手段,它以图形的形式描述软件系统中的模块组成和模块间的调用关系。构成系统结构图的主要成分有模块、调用和数据。在结构图中的模块以矩形表示,在矩形框内需要标示出模块的名字。模块之间的调用关系用箭头或直线表示。对于两个处于不同位置的模块,通常把位于上层的模块称为调用模块,把位于下层的模块称为被调用模块。表示调用关系的箭头或直线旁边的小箭头表示信息的传送:如果小箭头尾部带有空心圆,表示传送数据信息;如果小箭头尾部带有实心圆,表示传送控制信息。

【例 7-9】 从下列有关程序设计风格的叙述中选出 5 条正确的叙述。

- ① 使用括号以改善表达式的清晰性。
- ② 对递归定义的数据结构不要使用递归的过程。
- ③ 尽可能对程序代码进行优化。
- ④ 不要修补不好的程序,要重新写。
- ⑤ 不要进行浮点数的相等比较。
- ⑥ 应尽可能多输出中间结果。
- ⑦ 利用数据类型对数据值进行防范。
- ⑧ 用计数方法而不是用文件结束符来判别输入的结束。
- ⑨ 程序中的注释是可有可无的。
- ⑩ 使用有意义的标识符。

正确答案: ①、④、⑤、⑦、⑩。

分析: 好的程序设计风格,主要是指清晰易读、易于修改。正如好的文风、好的写作风格一样。好的程序设计风格应从多方面加以注意。对于复杂的表达式最好使用括号清楚地表明运算的优先次序;当数据结构是递归定义的时候最好使用递归的算法;应首先保证程序代码清晰、正确、可靠,不要片面追求“优化”和“技巧”;如果想要读懂不好的程序并将其修改好,比重新编写要费事得多;浮点数的运算是近似的,相等比较可

能永远不能达到；为保持程序的简洁性，应尽可能少输出中间结果；各种数据类型的数据运算结果都不相同，对数据值进行类型检查有助于保证运算的正确性；用计数方法比用文件结束符麻烦得多；完全没有注释的程序可能像天书一样难懂；使用有意义的标识符，有助于提高程序的可读性。

【例 7-10】最早体现结构化程序设计思想的程序设计语言是 A，最早使用 BNF 文法定义程序设计语言文法的语言是 B，最早提出类（class）的概念的语言是 C，最早完备地体现面向对象并提出继承概念的程序设计语言是 D，最早的人工智能语言是 E。

可选答案：

- A~E: ① Ada            ② Pascal            ③ ALGOL68            ④ ALGOL60  
         ⑤ SIMULA       ⑥ LISP            ⑦ PROLOG            ⑧ SMALLTALK  
         ⑨ C            ⑩ C++

正确答案：A: ②    B: ④    C: ⑤    D: ⑧    E: ⑥

分析：1970 年代初由 N.Wirth 提出的 Pascal 是最早体现结构化程序设计思想的程序设计语言，该语言的吸引力在于不牺牲语言的功能而具有明显的简洁性。它的特点是朴素而有效的一系列数据类型定义功能，用 Pascal 语言编写的程序具有良好的程序结构，有较好的可读性和可靠性。1958 年—1960 年由 ALGOL60 委员会推出的 ALGOL60 是第一个有较严格的语法定义的高级程序设计语言，在 ALGOL60 报告中，John Backus 和 Peter Naur 提出了一种用来描述语言文法的记号，叫做 Backus-Naur Form，简称 BNF。SIMULA 是 1960 年代末由 Dahl 等人在 ALGOL60 的基础上开发出来的，它是最早提出类概念的语言。在类概念中，一组数据和过程被当作一个单元进行处理，属于某类的对象生成后可以独立于生成这些对象的程序存在。类概念后来逐渐演化成数据抽象的概念，这对于面向对象程序设计语言的出现有很大影响。SMALLTALK 是 1970 年代初由 Alan Kay 在 SIMULA 的影响下开发出来的，它最早完备地体现了面向对象的思想。“面向对象”这一术语就来源于 SMALLTALK。面向对象程序设计语言的主要特征包括数据抽象、信息隐蔽、信息继承和动态绑定等。C++ 是目前商业上较为成功的面向对象程序设计语言。LISP 是最早的人工智能语言，是 1950 年代末由 Mccarthy 等设计的，主要用来处理符号表达式，它是以λ演算为基础的函数型程序设计语言。PROLOG 也是一种人工智能语言，是 1972 年由 A.Colmerauer 等基于 Horn 逻辑的逻辑型程序设计语言。此外，Ada 是 1983 年美国国防部推出的适用于嵌入式计算机系统的程序设计语言；C 是 1974 年 D.Ritchie 等推出的系统程序设计语言；ALGOL68 是 1963—1968 年期间由 ALGOL68 委员会设计的 ALGOL60 的后继语言。

【例 7-11】计算机程序设计语言经历了近 50 年的发展，出现了许多不同的程序设计语言。

例如，A 是低级语言，B 是高级语言，C 是描述性（说明性）语言，D 是面



向对象语言, E 是特别适宜在网络上运行的、可用于各种平台的一种面向对象程序设计语言。

可选答案:

- |              |         |             |         |
|--------------|---------|-------------|---------|
| A: ① PASCAL  | ② BASIC | ③ FORTRAN   | ④ 汇编语言  |
| B: ① 机器语言    | ② 汇编语言  | ③ FORTRAN   | ④ OS/2  |
| C: ① PASCAL  | ② C     | ③ FORTRAN   | ④ SQL   |
| D: ① FORTRAN | ② SQL   | ③ SMALLTALK | ④ COBOL |
| E: ① FORTRAN | ② C     | ③ Java      | ④ LISP  |

正确答案: A: ④ B: ③ C: ④ D: ③ E: ③

分析: PASCAL 语言和 FORTRAN 语言都是编译型语言, BASIC 语言是解释型语言, 只有汇编语言是一种与机器语言十分接近的语言, 它用助记符号表示机器指令中的操作码和操作数, 其书写形式在很大程度上取决于特定计算机的机器指令, 它是一种低级语言。

OS/2 是 IBM 公司推出的一种 32 位单用户多任务的操作系统, 不是程序设计语言。汇编语言和机器语言是低级语言, FORTRAN 语言则是一种高级语言。

SQL 语言是典型的关系数据库语言, 它是非过程的、描述性的语言, 它指示计算机做什么, 不用指明如何做。

Smalltalk 中引入了类和对象、继承和消息, 是典型的面向对象程序设计语言, 而 Java 则是由 SUN 公司开发的另一种面向对象的程序设计语言, 它的特点是可移植性好, 可用于 IBM PC、Macintosh 等各种平台, 特别适合于在网络上运行。

【例 7-12】一种最早用于科学计算的程序设计语言是 A; 一种提供指针和指针操作且不存在布尔类型的、应用广泛的系统程序设计语言是 B; 一种适合在互联网上编写程序可供不同平台上运行的面向对象程序设计语言是 C; 一种在解决人工智能问题上使用最多的、有强的表处理功能的函数程序设计语言是 D; 一种以谓词逻辑为基础的, 核心是事实、规则和推理机制的实用逻辑程序设计语言是 E。

可选答案:

- |               |           |             |          |
|---------------|-----------|-------------|----------|
| A~E: ① PASCAL | ② ADA     | ③ SMALLTALK | ④ SNOBOL |
| ⑤ C           | ⑥ ALGOL68 | ⑦ Java      | ⑧ LISP   |
| ⑨ PROLOG      | ⑩ FORTRAN |             |          |

正确答案: A: ⑩ B: ⑤ C: ⑦ D: ⑧ E: ⑨

分析: 程序设计语言从机器语言、汇编语言、高级语言到第四代语言, 它的发展越来越快, 功能越来越强。用语言编写的程序的可理解性也越来越接近人类的思维定式。

FORTRAN 语言是第一种用于科学计算的程序设计语言, 它出现于 20 世纪 50 年代中期。

C 语言是目前应用非常广泛的系统程序设计语言, 它提供了指针和指针操作且不存



在布尔类型。C 语言提供了很多底层函数调用,可执行与机器指令相关的操作,易用性好。

Java 语言是适应 Internet 发展的需要而产生的通用网络程序设计语言,它提供了更好的网络安全性和平台无关性,并采用面向对象编程;可扩充性强,是一种适合在互联网上编写程序可供不同平台上运行的面向对象程序设计语言。

LISP 语言是一种在解决人工智能问题上使用最多的、有强的表处理功能的函数程序设计语言;而 PROLOG 语言则是一种以谓词逻辑为基础的,借助于推理规则从已有事实推出新的事实的实用的逻辑程序设计语言。

【例 7-13】软件语言是用于书写计算机软件的语言。它主要包括需求定义语言、A、B、程序设计语言以及 C 等,适用于软件开发各个阶段。程序设计语言的基本成分是数据成分、运算成分、控制成分以及 D。程序设计语言有多种分类法,例如,按成分性质分,有顺序语言,并发语言,并行语言, E。

可选答案:

- A: ① 数据定义语言    ② 功能性语言    ③ 面向对象语言    ④ 函数式语言  
B: ① 设计性语言    ② 结构性语言    ③ 命令式语言    ④ 申述式语言  
C: ① 过程语言    ② 非过程语言    ③ 逻辑式语言    ④ 文档语言  
D: ① 对象成分    ② 变量成分    ③ 语句成分    ④ 传输成分  
E: ① 交互式语言    ② 分布语言    ③ 面向对象语言    ④ 高级语言

正确答案: A: ②    B: ①    C: ④    D: ④    E: ②。其中, A、B、C 的答案顺序可互换。

分析: 软件语言是用于书写计算机软件的语言。它主要包括需求定义语言、功能性语言、设计性语言、程序设计语言以及文档性语言等,适用于软件开发各个阶段。程序设计语言的基本成分是数据成分、运算成分、控制成分以及传输成分。程序设计语言有多种分类法,例如,按成分性质分,有顺序语言,并发语言,并行语言,分布语言。

【例 7-14】软件测试的目的是 A。为了提高测试的效率,应该 B。使用白盒测试方法时,确定测试数据应根据 C 和指定的覆盖标准。与设计测试数据无关的文档是 D。软件的集成测试工作最好由 E 承担,以提高集成测试的效果。

可选答案:

- A: ① 评价软件的质量    ② 发现软件的错误  
    ③ 找出软件中的所有错误    ④ 证明软件是正确的  
B: ① 随机地选取测试数据  
    ② 取一切可能的输入数据作为测试数据  
    ③ 在完成编码以后制定软件的测试计划  
    ④ 选择发现错误的的可能性大的数据作为测试数据  
C: ① 程序的内部逻辑    ② 程序的复杂程度

- ③ 使用说明书                      ④ 程序的功能  
D: ① 该软件的设计人员            ② 程序的复杂程度  
    ③ 源程序                        ④ 项目开发计划  
E: ① 该软件的设计人员            ② 该软件开发组的负责人  
    ③ 该软件的编程人员            ④ 不属于该软件开发组的软件人员  
正确答案: A: ②    B: ④    C: ①    D: ④    E: ④

分析: 软件测试的目的是发现软件中的错误。软件测试的一个最大的弱点是不可能把所有可能的输入数据都拿来测试(时间花费不起),因此不可能找出软件中的所有错误,所以软件测试只能表明软件中存在错误,不能证明软件是正确的。为了提高测试的效率,应该选择发现错误的可能性大的数据作为测试数据。

测试的方法分成白盒测试和黑盒测试两种。使用白盒测试方法时,确定测试数据应根据程序的内部逻辑和指定的覆盖标准,可以不考虑程序的功能,因此测试用例的设计仅与模块设计说明书及源程序有关,与设计测试数据无关的文档是项目开发计划。

软件的集成测试工作最好由不属于该软件开发组的软件设计人员承担,这样可以克服软件开发者过于自信的心理障碍,而且不同的人从不同角度看待软件开发过程及软件开发产品,能发现开发者不易发现的错误,也不会掩饰错误,可以提高集成测试的效果。

【例 7-15】 软件测试通常可分为单元测试、集成测试、确认测试和系统测试,其中确认测试主要用于发现 A 阶段的错误。在集成测试时,通常可采用自顶向下增殖式集成和自底向上增殖式集成。在自底向上增殖式集成时,对每个被集成的模块 B。对那些为众多用户开发的软件(如操作系统、编译程序),通常还要进行  $\alpha$  测试和  $\beta$  测试,以发现可能只有最终用户才能发现的错误。其中,  $\alpha$  测试是指最终用户在 C 的情况下所进行的测试,  $\beta$  是指最终用户在 D 的情况下所进行的测试。在软件维护阶段,当修改软件后,除了进行常规的测试外,还应进行 E 测试。

可选答案:

- A: ① 需求分析            ② 概要设计            ③ 详细设计            ④ 编码  
B: ① 不必设计驱动模块和桩(stub)模块  
    ② 不必设计驱动模块,但要设计桩模块  
    ③ 要设计驱动模块,但不必设计桩模块            ④ 要设计驱动模块和桩模块  
C: ① 开发环境下,开发人员不在场            ② 开发环境下,开发人员在场  
    ③ 用户的实际使用环境下,开发人员不在场  
    ④ 用户的实际使用环境下,开发人员在场  
D: ① 开发环境下,开发人员不在场            ② 开发环境下,开发人员在场  
    ③ 用户的实际使用环境下,开发人员不在场  
    ④ 用户的实际使用环境下,开发人员在场  
E: ① 恢复            ② 强度            ③ 安装            ④ 回归

正确答案: A: ① B: ③ C: ② D: ③ E: ③

分析: 软件测试通常可分为单元测试、集成测试、确认测试和系统测试, 其中确认测试主要用于发现软件需求分析阶段的错误。在集成测试时, 通常可采用自顶向下增殖式集成和自底向上增殖式集成。在自底向上增殖式集成时, 对每个被集成的模块要设计驱动模块, 但不必设计桩模块。对那些为众多用户开发的软件(如操作系统、编译程序), 通常还要进行 $\alpha$ 测试和 $\beta$ 测试, 以发现可能只有最终用户才能发现的错误。其中,  $\alpha$ 测试是指最终用户在开发环境下, 开发人员在场的情况下所进行的测试,  $\beta$ 是指最终用户在用户的实际使用环境下, 开发人员不在场的情况下所进行的测试。在软件维护阶段, 当修改软件后, 除了进行常规的测试外, 还应进行回归测试, 以检测原来的错误是否真正被克服。

【例 7-16】在设计测试用例时, A 是用得最多的一种黑盒测试方法。在黑盒测试方法中, 等价类划分方法设计测试用例的步骤是:

① 根据输入条件把数目极多的输入数据划分为若干个有效等价类和若干个无效等价类;

② 设计一个测试用例, 使其B覆盖尚未被覆盖的有效等价类, 重复这一步, 直至所有的有效等价类均被覆盖;

③ 设计一个测试用例, 使其覆盖C尚未被覆盖的无效等价类, 重复这一步, 直至所有的无效等价类均被覆盖;

因果图法是根据D之间的因果关系来设计测试用例的。

在实际应用中, 一旦纠正了程序中的错误后, 还应选择部分或全部原先已测试过的测试用例, 对修改后的程序重新测试, 这种测试称为E。

可选答案:

A: ① 等价类划分      ② 边值分析      ③ 因果图      ④ 判定表  
B~C: ① 1 个      ② 7 个左右      ③ 一半      ④ 尽可能少地  
⑤ 尽可能多地      ⑥ 全部

D: ① 输入与输出      ② 设计与实现      ③ 条件与结果      ④ 主程序与子程序

E: ① 验收测试      ② 强度测试      ③ 系统测试      ④ 回归测试

正确答案: A: ② B: ⑤ C: ① D: ① E: ④

分析: 等价类划分是典型的黑盒测试方法, 它将程序所有可能的输入数据按其发现错误的的能力划分到若干个等价类中, 再从每一个等价类中选取有代表性的输入数据作为测试数据。其选取测试数据的步骤分两步走: 第一步划分等价类, 第二步根据等价类选取测试用例。等价类分为两种, 一种是有效等价类, 它包含所有对于程序的规格说明来说是合理的预期的输入数据, 另一种是无效等价类, 它包含的输入数据对于程序的规格说明来说是不合理的非预期的。由于无效等价类能够比有效等价类发现更多的错误, 因此, 为有效发现程序中的错误, 对每一个无效等价类, 都需要设计一个测试用例。换言之

之,若一个测试数据中含有属于不同无效等价类的多个错误,有可能在测试中只发现一个错误,而其他错误将可能被忽视。所以在设计测试用例时,需要注意这样两个原则:

✧ 设计测试用例,使其尽可能多地覆盖尚未被覆盖的有效等价类;✧ 对于每一个尚未被覆盖的无效等价类,都要设计一个测试用例。

在黑盒测试方法中,用得最多的是边值分析方法,因为在输入或输出等价类的边界上最容易产生错误,所以选取测试用例时,取等价类边界上的数据比取等价类内部的数据,发现错误的能力更强。因果图法则是根据输入与输出之间的因果关系来设计测试用例的。

在实际应用中,一旦纠正了程序中的错误后,还应选择部分或全部原先已测试过的测试用例,对修改后的程序重新测试,这种测试称为回归测试。

【例 7-17】本流程图描述了某子程序的处理流程,现要求用白盒测试法对子程序进行测试。

根据判定覆盖、条件覆盖、判定/条件覆盖、多重条件覆盖(条件组合覆盖)、路径覆盖等五种覆盖标准,从供选择的答案中分别找出满足相应覆盖标准的最小的测试数据组(用①~⑩表示)。

可选答案:

- |                   |                   |                  |
|-------------------|-------------------|------------------|
| ① $a = 5, b = 1$  | ⑤ $a = 5, b = -1$ | ⑦ $a = 5, b = 1$ |
| ② $a = 5, b = -1$ | $a = -5, b = 1$   | $a = 0, b = 1$   |
| ③ $a = 5, b = 1$  | $a = -5, b = -1$  | $a = 0, b = -1$  |
| $a = -5, b = -1$  | ⑥ $a = 5, b = 1$  | $a = -5, b = 1$  |
| ④ $a = 5, b = 1$  | $a = 0, b = 0$    | ⑧ $a = 5, b = 1$ |
| $a = 0, b = -1$   | $a = -5, b = -1$  | $a = 0, b = -1$  |
| $a = -5, b = 1$   | $a = -5, b = -1$  | $a = 0, b = -1$  |
| $a = -5, b = -1$  | ⑩ $a = 5, b = 1$  | $a = -5, b = 1$  |
| ⑨ $a = 5, b = 1$  | $a = 5, b = 0$    | $a = -5, b = 0$  |
| $a = 0, b = -1$   | $a = 5, b = -1$   | $a = -5, b = -1$ |
| $a = 0, b = 1$    | $a = 0, b = 1$    |                  |
| $a = -5, b = 1$   | $a = 0, b = 0$    |                  |

正确答案:判定覆盖标准 ④

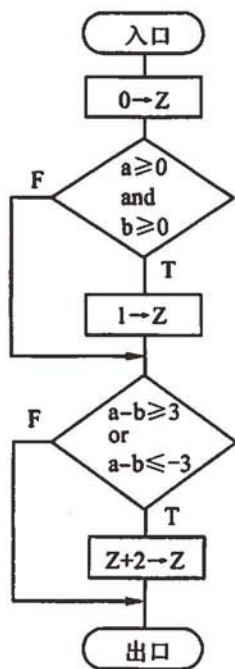
条件覆盖标准 ③

判定/条件覆盖标准 ⑥

多重条件覆盖标准 ⑧

路径覆盖标准 ⑦

分析:判定覆盖标准是指设计足够的测试用例,使得程序中每个判定的每一种可能结果都至少出现一次。本题中共有两个判定,每个判定的可能结果都只有两个:T和F,所以满足判定覆盖标准的测试数据至少有两个。取测试用例组④即满足要求:( $a = 5, b =$





1) 对于两个判定都为 T, ( $a=0, b=-1$ ) 对于两个判定都为 F。

条件覆盖标准是指设计足够的测试用例, 使得程序每一个判定中的每个条件的所有可能结果至少出现一次。在本题中, 判定内共有 4 个条件, 每个条件的可能结果都只有两个: T 和 F, 所以满足条件覆盖准则的测试数据至少有两个。设 ( $a \geq 0$ ) 取 T 为  $A_1$ , 取 F 为  $A_0$ ; ( $b \geq 0$ ) 取 T 为  $B_1$ , 取 F 为  $B_0$ ; ( $a-b \geq 3$ ) 取 T 为  $C_1$ , 取 F 为  $C_0$ ; ( $a-b \leq -3$ ) 取 T 为  $D_1$ , 取 F 为  $D_0$ 。若选测试用例组③即可满足要求。( $a=5, b=1$ ) 满足  $A_1B_1C_1D_0$ , ( $a=-5, b=-1$ ) 满足  $A_0B_0C_0D_1$ 。

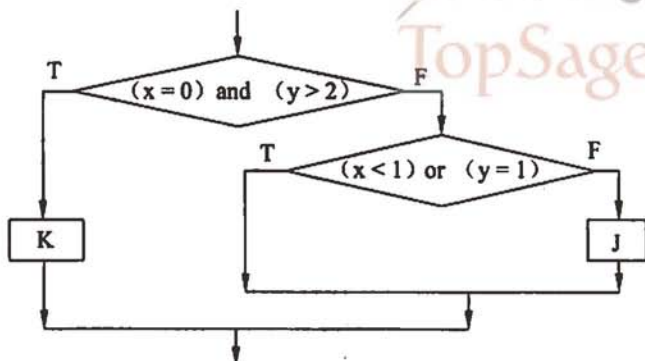
判定/条件覆盖标准是指设计足够的测试用例, 使得程序判定中每个条件的所有可能结果至少出现一次, 同时每个判定本身的所有可能结果至少出现一次。就是说, 满足判定/条件覆盖标准必须同时满足判定覆盖标准和条件覆盖标准。从原则上讲, 至少需要两个测试数据组, 但供选择的答案中给出的有两个测试数据组的答案都不能满足要求: ③满足条件覆盖标准, 不满足判定覆盖标准; ④满足判定覆盖标准, 不满足条件覆盖标准。因此, 需要选择有 3 个测试数据的组, 事实证明, 测试数据组⑥可满足要求。( $a=5, b=1$ ) 对于两个判定都为 T, 且满足  $A_1B_1C_1D_0$ ; ( $a=0, b=0$ ) 对于第一个判定为 T, 对于第二个判定为 F, 且满足  $A_1B_1C_0D_0$ ; ( $a=-5, b=-1$ ) 对于第一个判定为 F, 第二个判定为 T, 且满足  $A_0B_0C_0D_1$ 。

多重条件覆盖标准是指设计足够的测试用例, 使得程序每个判定中每个条件结果的所有可能组合至少经历一次。本题中每个判定各有两个条件, 因此各有四个条件取值的组合, 至少需要取四个测试数据组才能满足要求。取测试数据组⑧可满足多重条件覆盖标准。其中, ( $a=5, b=1$ ) 满足  $A_1B_1 \cdot C_1D_0$ ; ( $a=0, b=-1$ ) 满足  $A_1B_0 \cdot C_0D_0$ ; ( $a=-5, b=1$ ) 满足  $A_0B_1 \cdot C_0D_1$ ; ( $a=-5, b=-1$ ) 满足  $A_0B_0 \cdot C_0D_1$ 。注意, 第二个判定中的  $C_1D_1$  ( $a-b \geq 3 = T$  and  $a-b \leq -3 = T$ ) 不可能出现。

路径覆盖标准是指设计足够的测试用例, 使得程序中每条路径至少执行一次。在本题中共有四条路径, 至少应有四个测试数据组。若设第一个判定取真为  $P_1$ , 取假为  $P_0$ , 第二个判定取真为  $Q_1$ , 取假为  $Q_0$ 。不难分析出, 测试数据组⑦满足要求。( $a=5, b=1$ ) 满足  $P_1 \cdot Q_1$ ; ( $a=0, b=1$ ) 满足  $P_1 \cdot Q_0$ ; ( $a=0, b=-1$ ) 满足  $P_0 \cdot Q_0$ ; ( $a=-5, b=1$ ) 满足  $P_0 \cdot Q_1$ 。

**【例 7-18】** 在结构测试用例设计中, 有语句覆盖、条件覆盖、判定覆盖 (即分支覆盖)、路径覆盖等。其中, A 是最强的覆盖准则。为了对如下图所示的程序段进行覆盖测试, 必须适当地选取测试用例组。若  $x, y$  是两个变量, 可供选择的测试用例组共有 I、II、III、IV 四组 (如表中给出), 则实现判定覆盖至少应采用的测试用例组是 B 或 C; 实现条件覆盖至少应采用的测试用例组是 D; 实现路径覆盖至少应采用的测试用例组是 E 或 F。

	x	y
测试用例组 I	0	3
测试用例组 II	1	2
测试用例组 III	-1	2
测试用例组 IV	3	1



可选答案:

- A: ① 语句覆盖 ② 条件覆盖 ③ 判定覆盖 ④ 路径覆盖  
 B~E: ① I 和 II 组 ② II 和 III 组 ③ III 和 IV 组 ④ I 和 IV 组  
 ⑤ I、II 和 III 组 ⑥ II、III 和 IV 组 ⑦ I、III 和 IV 组 ⑧ I、II 和 IV 组

正确答案: A: ⑧ B: ⑤ C: ④ D: E: ⑤ F: ⑧。其中, A、B 的答案顺序可互换, E、F 的答案顺序可互换。

分析: 在结构测试中, 根据覆盖的目标不同, 可分为语句覆盖、条件覆盖、判定覆盖、路径覆盖等。其中路径覆盖是最强的覆盖准则。判定覆盖的含义是设计若干测试用例, 运行被测程序, 使得程序中每个判定的取真分支和取假分支至少执行一次; 条件覆盖的含义是设计若干测试用例, 运行被测程序, 使得程序中每个判定的每个条件的可能取值至少经历一次; 路径覆盖则要求覆盖程序中所有可能的路径。

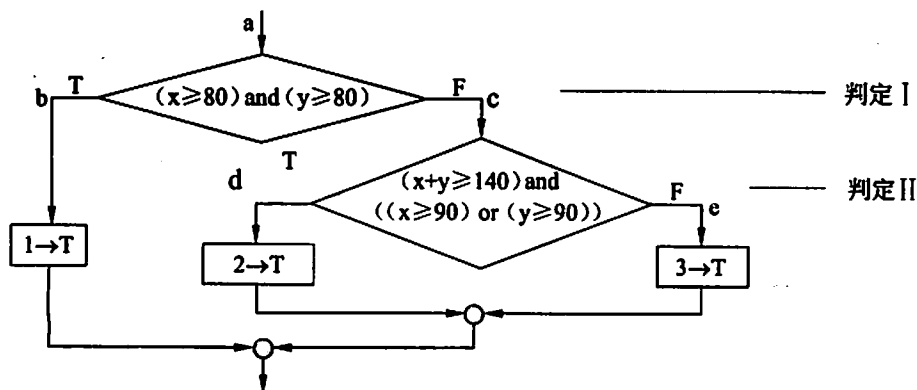
为判定覆盖选取测试用例情形: 对第一个判定选取测试用例组 I 和 II, 当用 I 覆盖判定的 T 分支时, 不会走到第二个分支; 当用 II 覆盖判定的 F 分支时, 第二个判定需另取一个测试用例组覆盖其 T 分支, 此时取测试用例组 III 或 IV 即可。

为条件覆盖选取测试用例情形: 取测试用例组 I 和 IV 就可以覆盖所有 4 个条件的取值。

为路径覆盖选取测试用例情形: 总共 3 条路径, 需 3 个测试用例, 可选使各路径为 T 的测试用例。I、II、III 或 I、II、IV 均可。

【例 7-19】本流程图描述了某子程序的处理流程, 现要求用白盒测试法对其进行测试。

如图所示的程序有三条不同的路径。分别表示为  $L_1 (a \rightarrow b)$ 、 $L_2 (a \rightarrow c \rightarrow d)$ 、 $L_3 (a \rightarrow c \rightarrow e)$ , 或简写为 ace、abd、abe 及 acd。根据判定覆盖、条件覆盖、判定—条件覆盖、条件组合覆盖和路径覆盖等五种覆盖标准, 从供选择的答案中分别找出满足相应覆盖标准的最小测试用例组。(用①~⑩回答)



可选答案:

- |                    |                    |                    |
|--------------------|--------------------|--------------------|
| ① $x = 90, y = 90$ | ⑥ $x = 90, y = 70$ | $x = 80, y = 70$   |
| ② $x = 50, y = 50$ | $x = 70, y = 90$   | $x = 90, y = 70$   |
| ③ $x = 90, y = 90$ | $x = 50, y = 50$   | $x = 70, y = 90$   |
| $x = 50, y = 50$   | ⑦ $x = 90, y = 90$ | ⑨ $x = 90, y = 90$ |
| ④ $x = 90, y = 70$ | $x = 50, y = 50$   | $x = 90, y = 70$   |
| $x = 40, x = 90$   | $x = 90, y = 70$   | $x = 90, y = 30$   |
| ⑤ $x = 90, y = 90$ | $x = 70, y = 90$   | $x = 70, y = 90$   |
| $x = 50, y = 50$   | ⑧ $x = 90, y = 90$ | $x = 30, y = 90$   |
| $x = 90, y = 70$   | $x = 50, y = 50$   | $x = 70, y = 70$   |
| $x = 50, y = 50$   | $x = 90, y = 70$   | $x = 30, y = 90$   |
| ⑩ $x = 90, y = 90$ | $x = 90, y = 30$   | $x = 70, y = 70$   |
| $x = 80, y = 80$   | $x = 70, y = 90$   | $x = 50, y = 50$   |

正确答案: 判定覆盖标准 ⑤ 条件覆盖标准 ⑧ 判定/条件覆盖标准 ⑧

多重条件覆盖标准 ⑨ 路径覆盖标准 ⑤

分析: 若设  $x \geq 80$  取真为  $A_1$ , 取假为  $A_0$ ;  $y \geq 80$  取真为  $B_1$ , 取假为  $B_0$ ;  $x + y \geq 140$  取真为  $C_1$ , 取假为  $C_0$ ;  $x \geq 90$  取真为  $D_1$ , 取假为  $D_0$ ;  $y \geq 90$  取真为  $E_1$ , 取假为  $E_0$ 。

针对覆盖标准, 相应的测试用例组如下 (表中加双划线的表示逻辑上不能参与判断)

判定覆盖 ⑤	$x = 90, y = 90$	判定 I 取 T
	$x = 50, y = 50$	判定 I 取 F, 判定 II 取 F
	$x = 90, y = 70$	判定 I 取 F, 判定 II 取 T
条件覆盖 ⑧	$x = 90, y = 90$	$A_1 * B_1, C_1 * (D_1 + E_1)$
	$x = 50, y = 50$	$A_0 * B_0, C_0 * (D_0 + E_0)$
	$x = 80, y = 70$	$A_1 * B_0, C_1 * (D_0 + E_0)$
	$x = 90, y = 70$	$A_1 * B_0, C_1 * (D_1 + E_0)$
	$x = 70, y = 90$	$A_0 * B_1, C_1 * (D_0 + E_1)$

判定一条件覆盖 ⑧	$x = 90, y = 90$	$A_1 * B_1, C_1 * (D_1 + E_0)$ , 判定 I 取 T
	$x = 50, y = 50$	$A_0 * B_0, C_0 * (D_0 + E_0)$ , 判定 I 取 F, 判定 II 取 F
	$x = 80, y = 70$	$A_1 * B_0, C_1 * (D_0 + E_0)$ , 判定 I 取 F, 判定 II 取 F
	$x = 90, y = 70$	$A_1 * B_0, C_1 * (D_1 + E_0)$ , 判定 I 取 F, 判定 II 取 F
	$x = 70, y = 90$	$A_0 * B_1, C_1 * (D_0 + E_1)$ , 判定 I 取 F, 判定 II 取 T
条件组合覆盖 ⑨	$x = 90, y = 90$	$A_1 * B_1, C_1 * (D_1 + E_1)$
	$x = 90, y = 70$	$A_1 * B_0, C_1 * (D_1 + E_0)$
	$x = 90, y = 30$	$A_1 * B_0, C_0 * (D_1 + E_0)$
	$x = 70, y = 90$	$A_0 * B_1, C_1 * (D_0 + E_1)$
	$x = 30, y = 90$	$A_0 * B_1, C_0 * (D_0 + E_1)$
	$x = 70, y = 70$	$A_0 * B_0, C_1 * (D_0 + E_0)$
	$x = 50, y = 50$	$A_0 * B_0, C_0 * (D_0 + E_0)$

在条件组合覆盖情形,  $(x \geq 90) \text{ or } (y \geq 90)$  的组合有 4 种, 与条件  $(x+y \geq 140)$  的组合应有 8 种, 但  $(x+y \geq 140=F) \text{ and } ((x \geq 90=T) \text{ or } (y \geq 90=T))$  不可能出现, 因此, 7 个测试用例就够了。

路径覆盖 ⑤	$x = 90, y = 90$	$A_1 * B_1, C_1 * (D_1 + E_1)$ , 覆盖路径 $a \rightarrow b$ , 执行 $1 \rightarrow T$
	$x = 50, y = 50$	$A_0 * B_0, C_0 * (D_0 + E_0)$ , 覆盖路径 $a \rightarrow c \rightarrow e$ , 执行 $3 \rightarrow T$
	$x = 90, y = 70$	$A_1 * B_0, C_1 * (D_1 + E_0)$ , 覆盖路径 $a \rightarrow c \rightarrow d$ , 执行 $2 \rightarrow T$

【例 7-20】在面向对象方法中, 对象可看成是属性(数据)以及这些属性上的专用操作的封装体。封装是一种 A 技术, 封装的目的是使对象的 B 分离。

类是一组具有相同属性和相同操作的对象的集合, 类中的每个对象都是这个类的一个 C。类之间共享属性和操作的机制成为 D。一个对象通过发送 E 来请求另一个对象为其服务。

可选答案:

- A: ① 组装                      ② 产品化                      ③ 固化                      ④ 信息隐蔽  
 B: ① 定义和实现              ② 设计和测试              ③ 设计和实现              ④ 分析和定义  
 C: ① 例证 (illustration)              ② 用例 (use case)  
     ③ 实例 (instance)                      ④ 例外 (exception)  
 D: ① 多态性                      ② 动态绑定                      ③ 静态绑定                      ④ 继承  
 E: ① 调用语句                      ② 消息                      ③ 命令                      ④ 口令

正确答案: A: ④    B: ①    C: ③    D: ④    E: ②

分析: 对象是面向对象开发模式的基本成份。每个对象可用它本身的一组属性和它可以执行的一组操作来定义。属性一般只能通过执行对象的操作来改变。操作又称为方法或服务, 在 C++ 中称为成员函数, 它描述了对象执行的功能。若通过消息传递, 还可以为其他对象使用。



“对象”有两个视图，分别表现在设计和实现方面。从设计方面来看，对象是一些概念的实例，它们把现实世界有关的实体模型化。这个视图把对象看做实体，产生有关实体的声明：描述实体，包括实体的属性和可以执行的操作，但不是描述实现功能的一系列动作。

从实现方面来看，一个对象封装了实体的实际数据结构和相应的操作的实现。对象是数据与操作的单一混合体。所谓封装，是一种信息隐蔽技术，其目的是把定义与实现分离，保护数据不被对象的使用者直接存取。

类是一组具有相同数据结构和相同操作的对象的集合。类的定义包括一组数据属性和在数据上的一组合法操作。类定义可以视为一个具有类似特性与共同行为的对象的模板，可用来产生对象。在一个类中，每个对象都是类的实例（Instance），它们都可使用类中提供的函数。概念的封装和实现的隐蔽，使得类具有更大的独立性。为便于类的调整，应尽量做到定义与实现分离。

【例 7-21】 OMT 是一种对象建模技术，它定义了三种模型，它们分别是 A 模型、B 模型和 C 模型。其中，A 模型描述了系统中对象的静态结构，以及对象之间的联系；B 模型描述系统中与时间和操作顺序有关的系统特征，表示瞬时的行为上的系统的“控制”特征，通常可用 D 来表示；C 模型描述了与值的变换有关的系统特征，通常可用 E 来表示。

可选答案：

- |         |       |       |        |
|---------|-------|-------|--------|
| A: ① 对象 | ② 功能  | ③ E-R | ④ 静态   |
| B: ① 控制 | ② 时序  | ③ 动态  | ④ 实时   |
| C: ① 对象 | ② 功能  | ③ 变换  | ④ 计算   |
| D: ① 类图 | ② 状态图 | ③ 对象图 | ④ 数据流图 |
| E: ① 类图 | ② 状态图 | ③ 对象图 | ④ 数据流图 |

正确答案：A: ④ B: ③ C: ② D: ③ E: ①

分析：Rumbaugh 等人提出对象模型技术（OMT），它把分析时收集的信息构造在三类模型中，即静态模型、功能模型和动态模型。静态模型的作用是描述系统的静态结构，包括构成系统的对象和类，它们的属性和操作，以及它们之间的联系。动态模型描述系统的控制逻辑，主要涉及系统中各个对象和类的时序及变化状况。动态模型包括两种图，即状态迁移图和事件追踪图。状态迁移图描述每一类对象的行为，事件追踪图描述发生于系统执行过程中的某一特定场景。功能模型着重于描述系统内部数据的传送与处理，它由多个数据流图组成。

静态模型是三个模型中最关键的一个模型。事实上，这个模型与实体-联系模型（E-R）十分相似。它可以不加限制地表示实体或联系的类型，各种以问题为中心的联系可以很自然地处理成继承联系。

动态模型着重于系统的控制逻辑。它包括两个图，一是状态图，一是事件追踪图。

状态图侧重于描述每一类对象的动态行为。事件追踪图侧重于说明发生于系统执行过程中的一个特定“场景”。场景是完成系统某个功能的一个事件序列。

功能模型着重于系统内部数据的传送和处理。它是模型化三角架的第三只脚。功能模型定义“做什么”，动态模型定义“何时做”，静态模型定义“对谁做”。功能模型表明，通过计算，从输入数据能得到什么样的输出数据。功能模型由多个数据流图组成。

【例 7-22】面向对象型的编程语言具有数据抽象、信息隐蔽、消息传递的 A 等特征。作为运算单位的对象应具有下列特性：B、C、D。E 是面向对象型的编程语言。

可选答案：

A: ① 对象调用 ② 对象变换 ③ 非过程性 ④ 信息继承 ⑤ 并发性

B~D: ① 对象把数据和处理数据的操作结合为一体

② 在程序运行时对象都处于活动状态

③ 对象在计算中可向其他对象发送消息

④ 接收消息的对象必须给消息发送者以回答

⑤ 对象的内部状态只根据外部送来的消息才操作

E: ① C++, SMALLTALK, object C ② C, Ada, Modula2

③ PASCAL, C++, APL ④ Ada, object C, C

正确答案: A: ⑤ B: ① C: ③ D: ⑤ E: ①

分析：面向对象的程序设计方法用对象和消息来描述事物和事物之间的关系。对象用数据描述自己的状态，同时规定对数据的操作，把数据和处理统一起来。对象总是作为一个整体使用，从外部只能看到其外部特性（能接收哪些消息、有哪些处理能力），看不到其内部特性（处理能力的实现、内部状态）。想要了解或改变其内部状态，使用其处理能力，都必须通过消息的传递。接受消息的对象可以返回信息，也可以不返回。发送者可以同时向多个对象传送消息；接受者可以同时接受多个对象的消息；对象之间也可以同时双向传送消息，这就是消息传递的并发性。

【例 7-23】A 是用于描述软件详细设计结果的语言，B 是支持动态绑定概念的语言，C 是支持强类型概念的语言，D 语言的一个主要特点是引用透明性，而在数据处理中使用的 E 是一种过程性语言。

可选答案：

A~E: ① PASCAL ② PDL ③ SMALLTALK80 ④ LISP ⑤ PSL

正确答案: A: ② B: ③ C: ① D: ④ E: ②

分析：PDL 是一种用于描述软件详细设计的语言。用 PDL 所描述的程序与用高级编程语言在总体结构上比较相似，其条件和处理可用自然语言书写。

绑定是指函数调用与执行该调用的程序代码之间建立联系，动态绑定是指在运行过程调用函数之前，与该调用相关联的程序代码是未知的，在执行调用时才发生关联。动态绑定是面向对象语言的重要特色，SMALLTALK80 语言具有这种功能。





执行路径,这可以视为单向的调用。

【例 7-25】 软件维护工作越来越受到重视,因为维护活动的花费常常要占软件生存周期全部花费的 A %左右。其工作内容为 B。为了减少维护工作的困难,可以考虑采取的措施为 C。而软件的可维护性包括 D, 所谓维护管理主要指的是 E。

可选答案:

- A: ① 10~20                      ② 20~40                      ③ 60~80                      ④ 90 以上
- B: ① 纠正和修改软件中含有的错误  
② 因环境发生变化,软件需做相应的变更  
③ 为扩充功能、提高性能而做的变更                      ④ 包括上述各点的内容
- C: ① 设法开发出无错误的软件                      ② 增加维护人员的数量  
③ 切实加强维护管理,并在开发过程中就采取有利于将来维护的措施  
④ 限制修改的范围
- D: ① 正确性、灵活性、可移植性                      ② 可测试性、可理解性、可修改性  
③ 可靠性、可复用性、可使用性                      ④ 灵活性、可靠性、高效性
- E: ① 加强需求分析                      ② 重新编码  
③ 判定修改的合理性并审查修改的质量                      ④ 加强维护人员的管理

正确答案: A: ③    B: ④    C: ③    D: ②    E: ③

分析: 软件维护活动所需工作量在整个软件生存期各个阶段中所占比重最大,大约占整个生存期工作量的 60~80%左右,这是因为在漫长的软件运行过程中需要不断对软件进行修改,以改正新发现的错误、适应新的环境和用户新的要求,这些修改需要花费很多精力和时间,而且有时修改不正确,还会引入新的错误。同时,软件维护技术不像开发技术那样成熟、规范化,自然消耗工作量就比较多。

软件维护主要有三方面的工作,即纠正和修改软件中含有的错误;为适应环境的变化而对软件做相应的变更;为扩充软件功能、提高软件性能而对软件做变更。

为了减少维护工作的困难,需要提高软件的可维护性,为此,需要做一些预防维护的工作,一是切实加强维护管理,二是在开发过程中就采取有利于将来维护的措施。

软件的可维护性对于软件的生存期至关重要,根据 Boehm 质量模型,软件可维护性包括可测试性、可理解性、可修改性。只要使软件具有这三种质量特性,软件就具有可维护性。

所谓维护管理主要指的是对软件修改的管理。判定修改的合理性并审查修改的质量,从而控制维护的成本和维护的效率。

【例 7-26】 软件可移植性是用来衡量软件 A 的重要尺度之一。为了提高软件的可移植性,应注意提高软件的 B。采用 C 有助于提高 B。为了提高可移植性,还应 D, 使用 E 语言开发的系统软件具有较好的可移植性。

可选答案:

- A: ① 通用性                      ② 效率                      ③ 质量                      ④ 人机界面



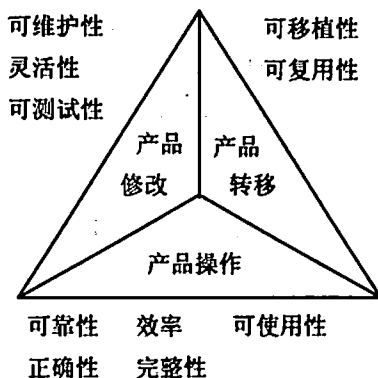
- B: ① 使用的方便性 ② 简洁性 ③ 可靠性 ④ 设备独立性  
 C: ① 优化算法 ② 专用设备 ③ 表格驱动方式 ④ 树形文件目录  
 D: ① 有完备的文档资料 ② 选择好的宿主计算机  
     ③ 减少输入输出次数 ④ 选择好的操作系统  
 E: ① COBOL ② APL ③ C ④ PL/I

正确答案: A: ③ B: ④ C: ③ D: ① E: ③

分析: 软件的可移植性是指把软件产品从一个硬件/软件环境转移到另一个硬件/软件环境的难易与繁简程度。它与可复用性、相互操作性(互连性)一起,从软件对新环境的适应性这一侧面,反映软件的质量。

为了提高软件的可移植性,应尽量使软件与具体设备无关,即提高软件的设备独立性,采用表格驱动的方式,有助于提高软件的设备独立性。为了便于软件移植,应有完备的文档资料。完全的能确切反映软件当前情况的文档,是十分重要的。它不仅有利于软件移植,而且有助于软件的顺利开发、可靠运行、全面测试、正确理解和使用。此外,可方便地维护、更新、修正、改动和扩充。

【例 7-27】软件质量包含多方面的内容, A、B、可移植性、可复用性等是较为重要的质量特性。在软件开发中,必须采取有力的措施以确保软件的质量,这些措施至少应包括: C、D、E。



可选答案:

- A~B: ① 稳定性 ② 可靠性 ③ 数据一致性 ④ 可维护性  
       ⑤ 可行性 ⑥ 数据独立性  
 C~E: ① 在开发初期制定质量保证计划,并在开发中坚持实行  
       ② 开发工作严格按阶段进行,文档工作应在开发完成后集中进行  
       ③ 严格执行阶段评审  
       ④ 要求用户参与全部开发过程,以监督开发质量  
       ⑤ 开发前选定或制定开发标准或开发规范,并遵照实施

## ⑥ 争取足够的开发经费和开发人力的支持

正确答案: A: ② B: ④ C: ① D: ③ E: ⑤。其中, A、B 的答案顺序可互换, C、D、E 的答案顺序可互换。

分析: 影响软件质量的因素很多。McCall 等人提出了一种软件质量特性分类的方法。这种分类方法着重考虑软件产品的三个侧面, 即软件的操作特性、软件易于移植的能力和适应新环境的能力。可靠性和可维护性分别是产品操作和产品转移这两个侧面中较重要的因素。软件的可靠性是指根据产品的功能和性能要求, 在给定的时间内, 程序成功运行的概率; 可维护性是指软件为适应新的环境和满足用户新的要求而实施变更的难易程度。

保证软件质量的措施至少应包含在软件开发初期指定质量保证计划, 并在开发过程中坚持实行; 严格执行阶段评审; 开发前选定或制定开发标准或开发规范, 并遵照实施等。

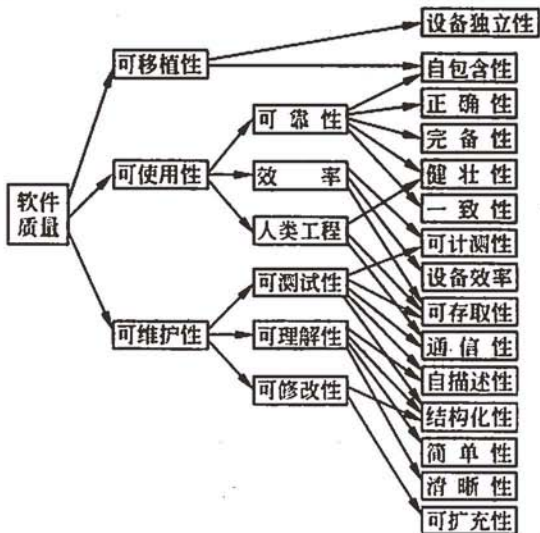
【例 7-28】根据 B.W.Boehm 提出的软件质量特征树模型, 软件质量应反映在效率、可靠性、人类工程、可维护性和 A 各个方面, 每一方面又层次式地相应取决于若干要素。比如, 软件的可维护性包括可修改性、B 及 C。其中 B 主要涉及到自描述性、清晰性、简单性、结构化性等, 而可修改性主要取决于 D 和 E。

可选答案:

- A~E: ① 可扩充性      ② 可移植性      ③ 完备性      ④ 可理解性  
⑤ 可测试性      ⑥ 健壮性      ⑦ 结构化性      ⑧ 设备独立性

正确答案: A: ② B: ④ C: ⑤ D: ⑦ E: ①。其中, D、E 的答案顺序可互换。

分析: Boehm 模型是 1976 年提出来的, 是早期的质量的分层模型。其后又出现影响很大的 McCall 模型及当前通用的 ISO9126 模型。Boehm 的质量特征树如下图所示。



【例 7-29】设计高质量的软件是软件设计追求的一个重要目标。可移植性、可靠性、可维护性、效率、可理解性和可使用性等都是评价软件质量的重要方面。

可移植性是反映出把一个原先在某种硬件或软件环境下正常运行的软件移植到另一个硬件或软件环境下,使该软件也能正常运行的难易程度。为了提高软件的可移植性,应注意提高软件的 A。可维护性通常包括 B。通常认为,软件维护工作包括改正性维护、C 维护和 D 维护。其中 C 维护是为了扩充原有软件的功能或提高原有软件的性能而进行的维护活动。

E 是指当系统万一遇到未预料的情况时,能够按预定的方式做合适的处理。

可选答案:

- A: ① 使用方便性    ② 简洁性    ③ 可靠性    ④ 设备不依赖性  
B: ① 可用性和可理解性    ② 可修改性、数据独立性和数据一致性  
③ 可测试性和稳定性    ④ 可理解性、可修改性和可测试性  
C~D: ① 功能性    ② 扩展性    ③ 合理性    ④ 完善性  
⑤ 合法性    ⑥ 适应性  
E: ① 可用性    ② 正确性    ③ 稳定性    ④ 健壮性

正确答案: A: ④    B: ④    C: ④    D: ⑥    E: ④

分析:软件的可移植性是指把程序从一种硬件配置和(或)软件系统环境转移到另一种配置或环境时需要的工作量的多少。提高软件可移植性的关键在于提高软件的设备无关性,即设备不依赖性。

软件的可维护性通常包括可理解性、可修改性和可测试性。按照每次维护的具体目标,软件维护工作可分为 3 类:改正性维护、完善性维护和适应性维护。改正性维护的目的是改正在开发期间未能发现而遗留下来的错误;完善性维护是指为了满足用户日益增长的需求,提高软件自身的市场竞争力而不断改善和加强产品功能和性能的维护活动;适应性维护则是指软件为适应运行环境的变化而进行的一种维护活动。

软件的健壮性是指在硬件发生故障、输入的数据无效或操作错误等意外时,即系统遇到未预料的情况时,系统能够做出适当响应的程度。

【例 7-30】国家标准《计算机软件产品开发文件编制指南 GB 8567-88》中规定,在一个软件项目的开发过程中,一般来说应产生 14 种文件。其中,管理人员主要使用的有 A、B、C、开发进度月报、项目开发总结报告。开发人员主要使用的有 A、C、D、数据要求说明书、概要设计说明书、详细设计说明书、数据库设计说明书、测试计划和 E。维护人员主要使用的有用户手册、C 和 E。

可选答案:

- A~E: ① 软件需求说明书    ② 项目开发计划    ③ 可行性研究报告  
④ 模块开发卷宗    ⑤ 测试分析报告    ⑥ 操作手册  
⑦ 用户手册

正确答案: A: ② B: ③ C: ④ D: ① E: ⑤

分析: 国家标准《计算机软件产品开发文件编制指南 GB 8567-88》中规定, 在一个软件项目的开发过程中, 一般来说应产生 14 种文件, 即项目开发计划、可行性研究报告、模块开发卷宗、开发进度月报、项目开发总结报告、软件需求说明书、数据要求说明书、概要设计说明书、详细设计说明书、数据库设计说明书、测试计划、测试分析报告、用户手册和操作手册。其中, 管理人员主要使用的有项目开发计划、可行性研究报告、模块开发卷宗、开发进度月报、项目开发总结报告。开发人员主要使用的有项目开发计划、可行性研究报告、软件需求说明书、数据要求说明书、概要设计说明书、详细设计说明书、数据库设计说明书、测试计划和测试分析报告。维护人员主要使用的有用户手册、测试分析报告和模块开发卷宗。

【例 7-31】通常, 软件开发环境可由环境机制和工具集构成。按功能划分, 环境机制又可分为 A; 工具集也可分为贯穿整个开发过程的工具和解决软件生存周期中某一阶段问题的工具, 分别属于上述两类的是 B。软件开发环境的核心是 C。软件开发环境具有集成性、开放性、D、数据格式一致性、风格统一的用户界面等特性, 因而能大幅度提高软件生产率。其中, 开放性是指 E。

可选答案:

- A: ① 环境操作系统、环境信息库、用户界面规范  
② 环境信息库、过程控制和消息服务、用户界面规范  
③ 环境操作系统、环境规格描述语言、过程控制和消息服务  
④ 环境规格描述语言、过程控制和消息服务、数据集成
- B: ① DFD、PDL ② HIPO 图、OOA ③ 文档管理工具、PAD 图  
④ 软件项目管理工具、软件价格模型及估算工具
- C: ① 环境操作系统 ② 环境信息库  
③ 环境规格描述语言 ④ 用户界面规范
- D: ① 可剪裁性 ② 完整性 ③ 封闭性 ④ 独立性
- E: ① 允许使用不同的硬件平台 ② 允许使用不同的操作系统  
③ 允许使用不同的网络系统 ④ 允许其他的软件工具加入到开发环境中

正确答案: A: ② B: ④ C: ② D: ① E: ④

分析: 本题的软件开发环境指的是集成型软件开发环境, 它是由环境机制和工具集构成的。环境继承机制主要有数据集成、控制集成和界面集成。数据集成指的是不同工具之间可以交换数据, 从技术上看, 最灵活的就是建立共享的环境信息库。控制集成用于支持环境中一个工具控制另一个工具, 即一个工具可以启动/终止某个工具或调用某个工具的特定服务, 通常利用消息传递手段达到这个目的。界面集成是指各工具采用相同的风格和共同的交互方式, 因此必须有用户界面规范。

工具集可分为贯穿整个开发过程的工具和解决软件生存周期中某一阶段问题的工



具,如软件项目管理工具就属于前者,而软件价格模型及估算工具属于后者。

软件开发环境的核心是共享的信息库,它提供接口给各个工具,一个工具的输出通过接口成为另一个工具的输入,实现了信息的共享。

软件开发环境具有集成性、开放性、可剪裁性、数据格式一致性、风格统一的用户界面等特性,其中开放性是指开发环境可以扩展,允许其他的软件工具加入到开发环境中。

## 7.3 思考练习题及答案

### 思考练习题

1. 软件是计算机系统中与硬件相互依存的另一部分,它是包括 A、B 及 C 的完整集合。其中, A 是按事先设计的功能和性能要求执行的指令序列。B 是使程序能够正确操纵信息的数据结构。C 是与程序开发、维护和使用有关的图文材料。

可选答案:

A~C: ① 软件      ② 程序      ③ 代码      ④ 硬件  
         ⑤ 文档      ⑥ 外设      ⑦ 数据      ⑧ 图表

2. 开发软件时对提高软件开发人员工作效率至关重要是 A。软件工程中描述生存周期的瀑布模型一般包括计划、B、设计、编码、测试、维护等几个阶段,其中设计阶段在管理上又可以依次分成 C 和 D 两步。

可选答案::

A: ① 程序开发环境      ② 操作系统的资源管理功能  
     ③ 程序人员数量      ④ 计算机的并行处理能力  
B: ① 需求分析      ② 需求调查      ③ 可行性分析      ④ 问题定义  
C~D: ① 方案设计      ② 代码设计      ③ 概要设计      ④ 数据设计  
         ⑤ 运行设计      ⑥ 详细设计      ⑦ 故障处理设计      ⑧ 软件体系结构设计

3. 软件工程的最终目的是以较少的投资获得可维护的、可靠的、高效率的和可理解的软件产品。软件工程技术应遵循 A、B、C、D、一致性、确定性、完备性、可验证性、抽象和信息隐蔽。

原型法适用于开发较复杂的系统,原型可分为三种: E、F 和 G。

可选答案:

A~D: ① 有效性      ② 合理性      ③ 局部化      ④ 协同性  
         ⑤ 实用性      ⑥ 模块化      ⑦ 抽象      ⑧ 信息隐蔽  
E~G: ① 复用型      ② 实验型      ③ 废弃型      ④ 演化型

## ⑤ 探索型

4. 当前系统的 A 模型描述现行系统的实际业务处理过程, 反映了现行系统具体 B 的现实。当前系统的 C 模型描述现行系统的功能结构、数据组织以及动态行为, 反映了现行系统 D 的本质。

目标系统是指待开发的新系统。根据计算机系统的特点, 分析、比较目标系统和当前系统逻辑上的差别, 确定目标系统的软件工作范围, 采用自顶向下逐步分解的分析策略, 确定目标系统的功能结构、数据组织以及动态行为, 从而建立起目标系统的 E 模型。

可选答案:

A, C, E: ① 对象      ② 物理      ③ 服务      ④ 过程      ⑤ 逻辑

B, D:    ① 怎么做    ② 何时做    ③ 做什么    ④ 为何做    ⑤ 谁来做

5. 软件需求分析方法必须能够理解和表达问题领域的信息域和功能域。信息域包括 A、B 和 C。

A 表示数据和控制传递时的变化方式。输入对象首先被变换成数据和控制的 D 信息, 然后再变换成输出结果信息。

B 表示信息在计算机中的组织形式。各种数据和控制对象按什么逻辑关系组织在一起, 又按什么物理关系存储在计算机中, 必须靠 B 分析来解决。

C 可以利用数据词典明确地表示, 也可以通过数据或数据对象的层次结构隐含地表示。

对数据进行变换就是程序所表现的功能。两个功能之间的数据传递确定了功能之间的 E。

可选答案:

A~C: ① 信息属性    ② 信息结构    ③ 信息服务    ④ 信息通信

⑤ 信息抽象    ⑥ 信息内容    ⑦ 信息流    ⑧ 信息层次

⑨ 信息项    ⑩ 信息行为

D~E: ① 连接    ② 接口    ③ 中间    ④ 通信    ⑤ 联系

6. 原型化方法是用户和软件开发人员之间进行的一种交互过程, 适用于 A 系统。它从用户界面的开发入手, 首先形成 B, 用户 C, 并就 D 提出意见, 它是一种 E 型的设计过程。

可选答案:

A: ① 需求不确定性高的    ② 需求确定的    ③ 管理信息    ④ 决策支持

B: ① 用户界面使用手册    ② 用户界面需求分析说明书

③ 系统界面原型    ④ 完善的用户界面

C: ① 改进用户界面的设计    ② 阅读文档资料

③ 模拟用户界面的运行      ④ 运行用户界面原型

D: ① 同意什么和不同意什么      ② 使用和不使用哪一种编程语言

③ 程序的结构      ④ 执行速度是否满足要求

E: ① 自外向内      ② 自顶向下      ③ 自内向外      ④ 自底向上

7. 块间联系和块内联系是评价程序模块结构质量的重要标准。联系的方式、共用信息的作用、共用信息的数量和接口的 A 等因素决定了块间联系的大小。在块内联系中, B 的块内联系最强。

SD 方法的总的原则是使每个模块执行 C 功能, 模块间传送 D 参数, 模块通过 E 语句调用其他模块, 而且模块间传送的参数应尽量 F。

此外, SD 方法还提出了判定的作用范围和模块的控制范围等概念。SD 方法认为, G 应该是 H 的子集。

可选答案:

A: ① 友好性      ② 健壮性      ③ 简单性      ④ 安全性

B: ① 巧合内聚      ② 功能内聚      ③ 通信内聚      ④ 信息内聚

C: ① 一个      ② 多个

D: ① 数据型      ② 控制型      ③ 混合型

E: ① 直接引用      ② 标准调用      ③ 中断      ④ 宏调用

F: ① 少      ② 多

G~H: ① 作用范围      ② 控制范围

8. 软件设计中划分程序模块通常遵循的原则是使各模块间的耦合尽可能 A。三种可能的模块耦合是:

B, 例如, 一个模块直接引用另一个模块中的数据。

C, 例如, 一个模块把开关量作为参数传送给另一个模块。

D, 例如, 一个模块通过公共数据结构把数据传送给另一个模块。

其中, E 的耦合性最强。(1991 年)

可选答案:

A: ① 强      ② 适中      ③ 弱

B~E: ① 公共耦合      ② 数据耦合      ③ 逻辑耦合      ④ 外部耦合

⑤ 内容耦合      ⑥ 控制耦合

9. 模块内聚度用于衡量模块内部各成分之间彼此结合的紧密程度。

(1) 一组语句在程序中多次出现, 为了节省内存空间, 把这些语句放在一个模块中, 该模块的内聚度是 A 的。

(2) 将几个逻辑上相似的成分放在一个模块中, 该模块的内聚度是 B。

(3) 模块中所有的成分引用共同的数据, 该模块的内聚度是 C。

(4) 模块内的某个成分的输出是另一些成分的输入, 该模块的内聚度是 D。

(5) 模块中所有成分结合起来完成一项任务, 该模块的内聚度是 E。它具有简明的外部界面, 由它构成的软件易于理解、测试和维护。

可选答案:

- A~E: ① 功能性      ② 顺序性      ③ 通信性      ④ 过程性  
         ⑤ 偶然性      ⑥ 瞬时性      ⑦ 逻辑性

10. 在众多的设计方法中, SD 方法是最受人注意的, 也是最广泛应用的一种, 这种方法可以同分析阶段的 A 方法及编程阶段的 B 方法前后衔接, SD 方法是考虑如何建立一个结构良好的程序结构, 它提出了评价模块结构质量的两个具体标准——块间联系和块内联系。SD 方法的最终目标是 C, 用于表示模块间调用关系的图叫 D。

另一种比较著名的设计方法是以信息隐蔽为原则划分模块, 这种方法叫 E 方法。

可选答案:

- A~B: ① Jackson      ② SA      ③ SC      ④ Parnas      ⑤ SP  
C: ① 块间联系大, 块内联系大      ② 块间联系大, 块内联系小  
     ③ 块间联系小, 块内联系大      ④ 块间联系小, 块内联系小  
D: ① PAD      ② HCP      ③ SC      ④ SADT  
     ⑤ HIPO      ⑥ NS  
E: ① Jackson      ② Parnas      ③ Turing      ④ Wirth      ⑤ Dijkstra

11. 软件详细设计工具可分为三类, 即: 图示工具、设计语言和表格工具。图示工具中, A 简单而应用广泛、B 表示法中, 每一个处理过程用一个盒子表示, 盒子可以嵌套。C 可以纵横延伸, 图形的空间效果好。

D 是一种设计和描述程序的语言, 它是一种面向 E 的语言。

可选答案:

- A~C: ① NS 图      ② 流程图      ③ HIPO 图      ④ PAD 图  
D: ① C      ② PDL      ③ RPOLOG      ④ PASCAL  
E: ① 人      ② 机器      ③ 数据结构      ④ 对象

12. 在编写程序时应采纳的原则之一是 A。开发软件时对提高开发人员工作效率至关重要是 B。软件工程中描述生存期的瀑布模型一般包括计划、C、设计、编码、测试、维护等几个阶段, 其中设计阶段又可依次分成 D 和 E 两步。

可选答案:

- A: ① 不限制 goto 语句的使用      ② 减少或取消注解行  
     ③ 程序越短越好      ④ 程序结构应有助于读者理解  
B: ① 操作系统的资源管理功能      ② 程序开发环境  
     ③ 程序人员数量      ④ 计算机的并行处理能力  
C: ① 需求分析      ② 需求调查      ③ 可行性分析      ④ 问题定义  
D~E: ① 数据结构设计      ② 详细设计      ③ 概要设计      ④ 数据库设计



## ⑤ 方案设计      ⑥ 代码设计

13. 1960 年底 Dijkstra 提倡的 A 是一种有效的提高程序设计效率的方法。

Dijkstra 为了使程序结构易于理解, 把基本控制结构限于顺序、B、C 3 种, 应避免使用 D。A 不仅提高程序设计的生产率, 同时也容易进行程序的 E。

可选答案:

A: ① 标准化程序设计      ② 模块化程序设计      ③ 多道程序设计      ④ 宏语言

⑤ 结构化程序设计      ⑥ 汇编语言      ⑦ 表格处理语言

B~C: ① 分支      ② 选择      ③ 重复      ④ 计算      ⑤ 输入输出

D: ① GOTO 语句      ② DO 语句      ③ IF 语句      ④ REPEAT 语句

E: ① 设计      ② 调试      ③ 维护      ④ 编码

14. 确定算法是解决问题的关键步骤之一。算法的工作量大小和实现算法所需的存储单元多少, 分别称为计算的 A 和 B。编写程序时, C 和 D 是应采纳的原则之一。E 是调试程序的主要工作之一。

可选答案:

A~B: ① 可实现性      ② 时间复杂度      ③ 空间复杂度

④ 困难度      ⑤ 高效性      ⑥ 计算有效性

C: ① 程序的结构化      ② 程序越短越好

③ 尽可能节省存储单元      ④ 尽可能减少注解行

D: ① 使用有实际意义的名字      ② 使用长度短并且无实际意义的名字

③ 表达式中尽量少用括号      ④ 尽量使用化简了的逻辑表达式

E: ① 调度      ② 证明程序正确      ③ 人员安排      ④ 纠错

15. 软件语言是指用于书写计算机软件的语言。它主要包括需求定义语言、功能性语言、设计性语言、程序设计语言和文档性语言等。A 就是一种典型的设计性语言, 常用于详细设计。B 语言是一种功能性语言, 它是以 C 理论为基础的一种规约语言。程序设计语言用于书写计算机程序, 它包含语法、语义和 D 三个方面。程序设计语言又可分为过程式语言和非过程式语言, 如 E 就是典型的非过程式语言。

可选答案:

A: ① PSL      ② PDL      ③ Eiffel      ④ Modula

B: ① Prolog      ② OBJ      ③ Java      ④ Z

C: ① 一阶谓词演算      ②  $\lambda$  演算      ③ 异调代数      ④ 范畴论

D: ① 语境      ② 语调      ③ 语用      ④ 词语

E: ① Prolog 和 RPG      ② Java 和 C++      ③ Lisp 和 Ada      ④ Java 和 Ada

16. A 在程序编码阶段进行, 它所依据的模块功能描述和内部细节以及测试方案应在 B 阶段完成, 目的是发现编写程序中的错误。

C 依据的模块说明书和测试方案应在 D 阶段完成, 它能发现设计错误。

E 应在模拟的环境中进行强度测试的基础上进行, 测试计划应在软件需求分析阶段制定。

可选答案:

- A: ① 用户界面测试    ② 输入输出测试    ③ 继承测试    ④ 单元测试  
B: ① 需求分析    ② 概要设计    ③ 详细设计    ④ 结构设计  
C: ① 集成测试    ② 系统性能测试    ③ 可靠性测试    ④ 强度测试  
D: ① 编写代码    ② 概要设计    ③ 维护    ④ 详细设计  
E: ① 过程测试    ② 函数测试    ③ 确认测试    ④ 逻辑路径测试

17. 软件测试是软件质量保证的主要手段之一, 测试的费用已超过 A 的 30% 以上, 因此提高测试的有效性非常重要。“高产”的测试是指 B。根据国家标准 GB 8566-88《计算机软件开发规范》的规定, 软件的开发和维护可划分为八个阶段, 其中, 单元测试是在 C 阶段完成的; 组装测试的计划是在 D 阶段制定的; 确认测试的计划是在 E 阶段制定的。

可选答案:

- A: ① 软件开发费用    ② 软件维护费用  
    ③ 软件开发和维护费用    ④ 软件研制费用  
B: ① 用适量的测试用例, 说明被测程序正确无误  
    ② 用适量的测试用例, 说明被测程序符合相应的要求  
    ③ 用适量的测试用例, 发现被测程序尽可能多的错误  
    ④ 用适量的测试用例, 纠正被测程序尽可能多的错误  
C~E: ① 可行性研究和计划    ② 需求分析  
      ③ 概要设计    ④ 详细设计    ⑤ 实现  
      ⑥ 组装测试    ⑦ 确认测试    ⑧ 使用和维护

18. 软件测试的目的是 A, 通常可分为白盒测试和黑盒测试。白盒测试根据程序的 B 来设计测试用例, 黑盒测试根据软件的规格说明来设计测试用例。常用的黑盒测试方法有边值分析、等价类划分、错误猜测、因果图等。其中, C 经常和其他方法结合起来使用。软件测试的步骤主要有单元测试、集成测试和确认测试。如果一个软件作为产品被许多客户使用的话, 在确认测试时通常要经过  $\alpha$  测试和  $\beta$  测试的过程。其中,  $\alpha$  测试是 D 进行的一种测试, 在软件设计与编码时, 采取 E 等措施都有利于提高软件的可测试性。

可选答案:

- A: ① 发现程序中的所有错误    ② 尽可能多地发现程序中的错误  
    ③ 证明程序是正确的    ④ 证明程序做了应做的事  
B: ① 功能    ② 性能    ③ 内部逻辑    ④ 内部数据  
C: ① 边值分析    ② 等价类划分    ③ 错误猜测    ④ 因果图

- D: ① 在开发者现场由开发方的非本项目的开发人员  
② 在开发者现场由用户  
③ 在用户现场由开发方的非本项目的开发人员  
④ 在用户现场由用户

- E: ① 不使用标准文本以外的语句, 书写详细正确的文档  
② 不使用标准文本以外的语句, 采用良好的程序结构  
③ 书写详细正确的文档, 信息隐蔽  
④ 书写详细正确的文档, 采用良好的程序结构

19. 从供选择的答案中选出同下列各条叙述关系最密切的字句。

- A. 对可靠性要求很高的软件, 例如操作系统, 由第三者对源代码进行逐行检查。  
B. 已有的软件被改版时, 由于受到变更的影响, 改版前正常的功能可能发生异常, 性能也可能下降。因此, 对变更的软件进行测试是必要的。  
C. 在意识到被测试模块的内部结构或算法的情况下进行测试。  
D. 为了确认用户的需求, 先做出系统的主要部分, 提交给用户试用。  
E. 在测试具有层次结构的大型软件时, 有一种方法是从上层模块开始, 由上到下进行测试。此时, 有必要用一些模块替代尚未测试过的下层模块。

可选答案:

- A~E: ① 仿真器      ② 代码审查      ③ 模拟器      ④ 桩  
⑤ 驱动器      ⑥ 域测试      ⑦ 黑盒测试      ⑧ 原型  
⑨ 白盒测试      ⑩ 退化测试

20. 软件测试方法可分为黑盒测试法和白盒测试法两种。

黑盒测试法是通过分析程序的 A 来设计测试用例的方法。除了测试程序外, 它还适用于对 B 阶段的软件文档进行测试。

白盒测试法是根据程序的 C 来设计测试用例的方法。除了测试程序外, 它也适用于对 D 阶段的软件文档进行测试。

白盒法测试程序时常按照给定的覆盖条件选取测试用例。E 覆盖比 F 覆盖严格, 它使得每一个判定的每一条分支至少经历一次。G 覆盖既是判定覆盖, 又是条件覆盖, 但它并不保证使各种条件都能取到所有可能的值。H 覆盖比其他条件都要严格, 但它不能保证覆盖程序中的每一条路径。

单元测试一般以 I 为主, 测试的依据是 J。

可选答案:

- A, C: ① 应用范围      ② 内部逻辑      ③ 功能      ④ 输入数据  
B~D: ① 编码      ② 软件详细设计      ③ 软件总体设计      ④ 需求分析  
E~H: ① 语句      ② 判定      ③ 条件      ④ 判定/条件  
⑤ 多重条件      ⑥ 路径

I: ① 白盒法                      ② 黑盒法

J: ① 模块功能规格说明    ② 系统模块结构图    ③ 系统需求规格说明

21. 集成测试也叫做 A 或 B。通常, 在 C 的基础上, 将所有模块按照设计要求组装成为系统。子系统的集成测试特别称为 D, 它所做的工作是要找出子系统和系统需求规格说明之间的 E。需要考虑的问题是: 在把各个模块        连接起来的时候, 穿越模块接口的数据是否会 F; 一个模块的功能是否会对另        一个模块的功能产生不利的影响; 各个 G 组合起来, 能否达到预期要求的 H; I 是否有问题: 单个模块的误差累积起来是否会放大。

可选答案:

A~D: ① 单元测试      ② 部件测试      ③ 组装测试      ④ 系统测试  
⑤ 确认测试      ⑥ 联合测试

E~I: ① 子功能          ② 丢失          ③ 父功能          ④ 局部数据结构  
⑤ 全局数据结构 ⑥ 不一致      ⑦ 一致

22. 软件测试中常用的静态分析方法是 A 和 B。B 用于检查模块或子程序间的调用是否正确。分析方法(白盒方法)中常用的方法是 C 方法。非分析方法(黑盒方法)中常用的方法是 D 方法和 E 方法。E 方法根据输出对输入的依赖关系设计测试用例。

可选答案:

A~B: ① 引用分析          ② 算法分析          ③ 可靠性分析      ④ 效率分析  
⑤ 接口分析          ⑥ 操作分析

C~E: ① 路径测试          ② 等价类          ③ 因果图          ④ 归纳测试  
⑤ 综合测试          ⑥ 追踪          ⑦ 深度优先      ⑧ 调试  
⑨ 相对图

23. 从下列关于软件测试的叙述中, 选出 5 条正确的叙述。

- ① 用黑盒法测试时, 测试用例是根据程序内部逻辑设计的。
- ② 尽量用公共过程或子程序去代替重复的代码段。
- ③ 测试是为了验证该软件已正确地实现了用户的要求。
- ④ 对于连锁型分支结构, 若有  $n$  个判定语句, 则有  $2^n$  条路径。
- ⑤ 尽量采用复合的条件测试, 以避免嵌套的分支结构。
- ⑥ GOTO 语句概念简单, 使用方便, 在某些情况下, 保留 GOTO 语句反能使写出的程序更加简洁。
- ⑦ 发现错误多的程序模块, 残留在模块中的错误也多。
- ⑧ 黑盒测试方法中最有效的是因果图法。
- ⑨ 在做程序的单元测试时, 桩(存根)模块比驱动模块容易编写。
- ⑩ 程序效率的提高主要应通过选择高效的算法来实现。



24. 测试大型软件通常由 A、集成测试、确认测试组成。确认测试主要寻找与软件 B 说明不一致的错误。语句覆盖、判定覆盖、条件覆盖和路径覆盖等都是用白盒测试法设计测试用例的覆盖准则。在这些覆盖准则中, 最弱的准则是 C, 最强的准则是 D。此外, 还有多种黑盒测试的设计测试用例的方法, 如 E。

可选答案:

- A:      ① 组装测试      ② 性能测试      ③ 接口测试      ④ 单元测试  
B:      ① 需求规格      ② 概要设计      ③ 详细设计      ④ 界面设计  
C~D:    ① 语句覆盖      ② 条件覆盖      ③ 路径覆盖      ④ 判定覆盖  
E:      ① E-R 图      ② 因果图      ③ DFD 图      ④ IPO 图

25. 对象是面向对象开发模式的 A。每个对象可用它自己的一组 B 和它可以执行的一组 C 来表征。应用执行对象的 C 可以改变该对象的 B。它的应用必须通过 D 的传递。可以认为, 这种 D 的传递大致等价于过程性范型中的函数调用。某些语言提供了特殊功能, 允许对象引用自己。若一个对象没有显式地被引用, 则可让该对象 E。

可选答案:

- A:      ① 基本单位      ② 最小单位      ③ 最大单位      ④ 语法单位  
B~C:    ① 行为      ② 功能      ③ 操作      ④ 数据      ⑤ 属性  
D:      ① 接口      ② 消息      ③ 信息      ④ 操作      ⑤ 过程  
E:      ① 撤销      ② 歇着      ③ 默认      ④ 隐式引用  
⑤ 引用自己

26. 类常常被看做是一个抽象数据类型的实现, 更合适的是把类看做是某种 A 的一个模型。事实上, 类是单个的 B 语义单元。类的用户能够操纵的操作叫做类的 C。类定义的其余部分给出数据定义和辅助功能定义, 包括类的实现。

类的实现常常包括了其他类的实例, 这些实例 D 被其他对象存取, 包括同一个类的其他实例。类的实现可能还包括某些私有方法, 实现它们的类可以使用, 而其他任何对象都不能使用。

类, 就它是一个数据值的聚合的意义上来看, 与 Pascal 中的记录或 C 中的结构类似, 但又有差别。类扩展了通常的记录语义, 可提供各种级别的 E。类不同于记录, 因为它们包括了操作的定义, 这些操作与类中声明的数据值有相同的地位。

可选答案:

- A:    ① 功能      ② 概念      ③ 结构      ④ 数据  
B:    ① 语法      ② 词法      ③ 语义      ④ 上下文环境  
C:    ① 界面      ② 操作      ③ 行为      ④ 活动  
D:    ① 可自由地      ② 可有控制地      ③ 可通过继承      ④ 应受保护不  
E:    ① 可移植性      ② 可重复性      ③ 可访问性      ④ 继承性

27. 在 C++ 语言中引进了类的概念。类的定义包括类名、类的说明和类的实现。 A

是类的外部接口，B 是类的内部表示，类具有 C、D 和 E。

有了 C 可以隐藏类对象内部实现的复杂细节，有效地保护内部所有数据不受外部破坏；D 增强了类的共享机制，实现了类的可复用性，简化系统的开发工作；E 可实现函数重载和运算符重载。

可选答案：

- A~B: ① 类的引用      ② 类的说明      ③ 类的实现      ④ 类的标识  
          ⑤ 类的构造      ⑥ 类的成员说明
- C~E: ① 开放性      ② 封装性      ③ 兼容性      ④ 继承性  
          ⑤ 多态性      ⑥ 可扩充性

28. 一个软件产品开发完成投入使用后，常常由于各种原因需要对它做适当的变更。在软件的使用过程中，软件原来的 A 可能不再适应用户的要求，需要进行变更；软件的工作环境也可能发生变化，最常见的是配合软件工作的 B 有变动；还有一种情况是在软件使用过程中发现错误，需要进行修正。通常把软件交付使用后做的变更称为 C。软件投入使用后的另一项工作是 D，针对这类软件实施的软件工程活动，主要是对其重新实现，使其具有更好的 E，包括软件重构、重写文档等。D 和新的软件开发工作的主要差别在于 (H)。我们把常规的软件开发称为 F，而 G 是从代码开始推导出设计或是规格说明来。

可选答案：

- A~B:      ① 环境      ② 软件      ③ 硬件  
          ④ 功能和性能      ⑤ 要求
- C, D, F, G: ① 逆向工程      ② 正向工程      ③ 软件再工程      ④ 维护  
          ⑤ 设计
- E:      ① 可靠性      ② 可维护性      ③ 可移植性      ④ 可修改性
- H:      ① 使用的工具不同      ② 开发的过程不同  
          ③ 开发的起点不同      ④ 要求不同

29. 在软件维护的实施过程中，为了正确、有效地修改，需要经历以下 3 个步骤：A、B、C。A 是决定维护成败和质量好坏的关键。C 包括 D 确认、计算机确认和维护后的 E。

可选答案：

- A~C: ① 修改程序      ② 建立目标程序      ③ 分析和理解程序  
          ④ 重新验证程序      ⑤ 验收程序
- D:      ① 动态      ② 静态      ③ 人工      ④ 自动
- E:      ① 验证      ② 验收      ③ 检验      ④ 存档

30. 下面有关软件维护的叙述有些是不准确的，请将它们列举出来。

- ① 要维护一个软件，必须先理解这个软件。

- ② 阅读别人写的程序并不困难。
- ③ 如果文档不齐全也可以维护一个软件。
- ④ 谁写的软件就得由谁来维护这个软件。
- ⑤ 设计软件时就应考虑到将来的可修改性。
- ⑥ 维护软件时一件很吸引人的创造性工作。
- ⑦ 维护软件就是改正软件中的错误。
- ⑧ 维护好一个软件是一件很难的事情。

31. 软件的易维护性是指理解、改正、改进软件的难易程度。通常影响软件易维护性的有易理解性、易修改性和 A。在软件的开发过程中往往采取各种措施来提高软件的易维护性, 如采用 B 有助于提高软件的易理解性; C 有助于提高软件的易修改性。

在软件质量特性中, D 是指在规定的条件和时间内, 与软件维持其性能水平的能力有关的一组属性; E 是指防止对程序及数据的非授权访问的能力。

可选答案:

- |            |          |           |         |
|------------|----------|-----------|---------|
| A: ① 易使用性  | ② 易恢复性   | ③ 易替换性    | ④ 易测试性  |
| B: ① 增强健壮性 | ② 信息隐蔽原则 | ③ 良好的编程风格 | ④ 高效的算法 |
| C: ① 高效的算法 | ② 信息隐蔽原则 | ③ 增强健壮性   | ④ 身份认证  |
| D: ① 正确性   | ② 准确性    | ③ 可靠性     | ④ 易使用性  |
| E: ① 安全性   | ② 适应性    | ③ 灵活性     | ④ 容错性   |

32. 从下列叙述中选出 5 条与提高软件的可移植性有关的叙述。

- ① 把程序中与计算机硬件特性有关的部分集成在一起。
- ② 选择时间效率和空间效率高的算法。
- ③ 使用结构化的程序设计方法。
- ④ 尽量用高级语言编写程序中对效率要求不高的部分。
- ⑤ 尽可能减少注释。
- ⑥ 采用表格控制方式。
- ⑦ 文档资料详尽、正确。
- ⑧ 在有虚拟存储器的计算机系统上开发软件。
- ⑨ 减少程序中对文件的读写次数。
- ⑩ 充分利用宿主计算机的硬件特性。

33. 估算资源、成本和进度时需要经验、有用的历史信息、足够的定量数据和作定量度量的勇气。通常估算本身带有 A。项目的复杂性越高, 规模越大, 开发工作量 B, 估算的 A 就 C。项目的结构化程度提高, 进行精确估算的能力就能 D, 而风险将 E。有用的历史信息 F, 总的风险会减少。

可选答案:

- |              |      |      |      |
|--------------|------|------|------|
| A: ① 风范 (范型) | ② 风格 | ③ 风险 | ④ 度量 |
|--------------|------|------|------|

- B~F: ① 增加 ② 越多 ③ 降低 ④ 不变  
⑤ 越少 ⑥ 越高 ⑦ 越大

34. 软件项目的进度管理有许多方法, 但 A 不是常用的进度控制图示方法。在几种进度控制的方法中, B 难以表达多个子任务之间的逻辑关系, 使用 C 不仅能表达子任务间的依赖关系, 还可找出关键子任务。在 C 中, 箭头表示 D, 圆圈结点表示 E。

可选答案:

- A~C: ① 甘特图 ② IPO ③ PERT ④ 时标网状图  
D~E: ① 数据流 ② 控制流 ③ 事件 ④ 活动  
⑤ 数据终点 ⑥ 数据源点

35. 任何软件项目都必须做好项目管理工作, 最常使用的进度管理工具是 A, 当某一开发项目的进度有可能拖延时, 应该 B。对于一个典型的软件开发项目, 各个开发阶段需投入的工作量的百分比大致是 C。各个阶段所需不同层次的技术人员大致是 D, 而管理人员在各阶段所需数量也不同, 相对而言大致是 E。

可选答案:

- A: ① 数据流图 ② 程序结构图 ③ 因果图 ④ PERT 图  
B: ① 增加新的开发人员 ② 分析拖期原因加以补救  
③ 从别的小组抽调人员临时帮忙 ④ 推迟预定完成时间

		需求分析	设计	编码	测试
C: 投入工作量	①	25	25	25	25
	②	10	20	30	40
	③	15	30	15	40
	④	5	10	65	30
D: 技术人员水平	①	初级	高级	高级	高级
	②	中级	中级	高级	中级
	③	高级	中高级	初级	中高级
	④	中级	中高级	中级	初级
E: 管理人员数量	①	多	中	少	中
	②	中	中	中	中
	③	多	少	多	多
	④	少	多	少	多

36. 风险分析实际上是 4 个不同的活动, 按顺序依次为 A、B、风险评价和 C。在风险评价时, 应当建立一个三元组:  $[r_i, l_i, x_i]$ ;  $r_i$  是风险描述,  $l_i$  是 D, 而  $x_i$  是风险的影响。一个对风险评价很有用的技术是定义 E。F、G、(H) 是三种典型的 E。在做风险分析的上下文环境中一个 E 就存在一个单独的点, 叫做参照点或 I。在这个



点上要公正地给出判断。实际上,参照点能在图上表示成一条平滑的曲线的情况很少,多数情况它是一个 J。

可选答案:

- A~C: ① 风险驾驭和监控 ② 风险识别 ③ 风险估计 ④ 风险消除  
 D: ① 风险的大小 ② 风险的概率 ③ 风险的时间  
 ④ 风险的范围  
 E: ① 风险参照水准 ② 风险度量 ③ 风险监控 ④ 风险工具  
 F~H: ① 生产率 ② 功能 ③ 成本 ④ 进度  
 ⑤ 范围 ⑥ 性能  
 I~J: ① 凹点 ② 崩溃点 ③ 终点 ④ 区域  
 ⑤ 拐点 ⑥ 原点

37. 软件项目的进度管理有许多方法,但 A 不是常用的进度控制图示方法。在几种进度控制图示方法中, B 难以表达多个子任务之间的逻辑关系,使用 C 不仅能表达子任务之间的逻辑关系,而且可以找出关键子任务。在 C 中,用带箭头的边表示 D,用圆圈结点表示 E,它标明 D 的 F。

可选答案:

- A~C: ① 甘特图 ② IPO ③ PERT ④ 时标网状图  
 D~F: ① 数据流 ② 控制流 ③ 事件 ④ 处理  
 ⑤ 起点或终点 ⑥ 任务

38. 软件项目管理的主要职能包括: A, 建立组织, 配备人员, B 和 C。由于软件项目的特有性质,使得项目管理存在一定困难。第一、D, 软件工程过程充满了大量高强度的脑力劳动;第二、E, 在特定机型上,利用特定的硬件配置,由特定的系统软件和支撑软件支持,形成了特定的开发环境;第三、F, 软件项目经历的各个阶段都深透了大量的手工劳动,远未达到自动化的程度;第四、G, 用户要经过专门的培训,才能掌握操作步骤,且需要配备专职维护人员进行售后服务;第五、H, 为高质量地完成软件项目,充分发掘人员的智力才能和创造精神。

在总结和分析足够数量失误的软件项目之后可知,造成软件失误的原因大多与 I 工作有关。在软件项目开始执行时,执行的过程中及项目进行的最后阶段都会遇到种种问题。

可选答案:

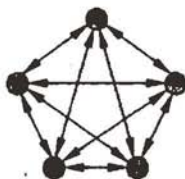
- A~C: ① 编码 ② 制定计划 ③ 开发 ④ 指导  
 ⑤ 测试 ⑥ 检验  
 D~H: ① 软件工作渗透了人的因素 ② 智力密集,可见性差  
 ③ 单件生产 ④ 使用方法繁琐,维护困难  
 ⑤ 劳动密集,自动化程度低

- I:      ① 设计                      ② 维护                      ③ 测试                      ④ 管理  
          ⑤ 实践                      ⑥ 指导                      ⑦ 审核                      ⑧ 分析

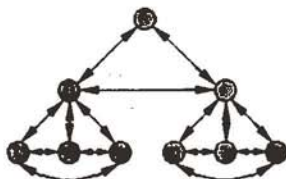
39. 软件开发小组的目的是发挥集体的力量进行软件研制。因此, 小组从培养 A 的观点出发进行程序设计消除软件的 B 的性质。通常, 程序设计小组的组织形式有三种, 如下图所示的 a 属于 C, b 属于 D, c 属于 E。



(a)



(b)



(c)

可选答案:

A~B:    ① “局部”                      ② “全局”                      ③ “集体”                      ④ “个人”

C~E:    ① 层次式小组                      ② 民主制小组                      ③ 主程序员制小组

40. 如何评价软件的质量一直是软件技术人员所关心的问题。目前已有多种软件质量模型来描述软件的质量特性。ISO/IEC 9126 是国际标准化组织在 1991 年提出的软件质量标准, 它由三个层次组成。第一层是质量特性, 第二层是质量子特性, 第三层是度量指标。六个质量特性是: 功能性、可靠性、易使用性、效率、可维护性和 A。其中, 功能性包括质量子特性 B; 可靠性包括质量子特性 C; 易使用性包括质量子特性 D; 可维护性包括质量子特性 E。

供选择的答案

- A: ① 易理解性                      ② 容错性                      ③ 可移植性                      ④ 安全性  
 B: ① 互用性                      ② 易恢复性                      ③ 易安装性                      ④ 易替换性  
 C: ① 依从性                      ② 易恢复性                      ③ 资源特性                      ④ 一致性  
 D: ① 易理解性                      ② 容错性                      ③ 易分析性                      ④ 安全性  
 E: ① 准确性                      ② 容错性                      ③ 易操作性                      ④ 易测试性

41. 选择对应关系题

(1) 在软件开发中以下几方面的内容应分别在哪个文档中得到阐明:

- A. 软件总体结构  
 B. 运行环境  
 C. 出错处理设计

(2) 以下两个文档应分别在哪两个阶段中开发:

- D. 初步用户手册  
 E. 确认测试计划

可选答案:

- A~C: ① 可行性研究报告      ② 项目开发计划      ③ 软件需求说明书  
④ 数据要求说明书      ⑤ 概要设计说明书      ⑥ 详细设计说明书  
⑦ 测试计划      ⑧ 测试报告      ⑨ 用户手册
- D~E: ① 可行性研究与计划      ② 需求分析      ③ 概要设计  
④ 详细设计      ⑤ 软件测试      ⑥ 软件维护

42. 从下列关于文档编制的叙述中选出五条正确的叙述。

① 可行性研究报告应评述为了合理地达到开发目标而可能选择的各种方案, 以便用户抉择。因此, 编写者不必提出结论。

② 操作手册的编写工作应该在软件测试阶段之前完成。

③ 软件的开发单位应该建立本单位文档的标识方法, 使文档的每一页都具有明确的标识。

④ 为了使得文档便于修改保持一致性, 各文档的内容不应有相互重复的地方。

⑤ 用户手册要使用专门术语, 并充分地描述该软件系统的结构及使用方法。

⑥ 详细设计说明书中可以使用判定表及必要的说明来表示程序的逻辑。

⑦ 概要设计说明书中可以使用 IPO 图来说明接口设计。

⑧ 测试分析报告应把每个模块实际测试的结果, 与软件需求规格说明书和概要设计说明书中规定的要求进行对照并作出结论。

⑨ 软件需求规格说明书中可以对软件的操作人员和维护人员的教育水平和技术专长提出要求。

⑩ 项目开发计划除去规定项目开发所需的资源、开发的进度等以外, 还可以包括用户培训计划。

43. 按制定软件工程标准的不同层次和适用范围, 软件工程标准可分为五级, A 制定的是国际标准, B 制定的是行业标准。GB1526-89 对程序流程图等作了明确具体的规定, 这种标准程序流程图的特点有 C、D、E。

可选答案:

- A~B: ① IEEE, GJB      ② IEEE, ANSI      ③ ISO, IEC      ④ ISO, IEEE  
⑤ IEC, GJB      ⑥ ANSI, ISO

- C~E: ① 箭头表示数据的传递方向  
② 允许自行定义多种特定的符号  
③ 对循环结构定义了一对特定的符号  
④ 它与 ISO 的有关规定有一些差别  
⑤ 允许存在具有两个以上处理的判定  
⑥ 特定方向的流线才用箭头标明流向  
⑦ 不允许在图形符号外加注标识符和描述符

44. 用于辅助软件开发、运行、维护、管理、支持等过程中的活动的软件称为软件开发工具，通常也称为 A 工具。

使用软件开发工具有助于提高软件的开发、维护和管理效率。集成型软件开发环境通常由工具集和环境集成机制组成。这种环境应具有 B，环境集成机制主要有数据集成机制、控制集成机制和界面集成机制。

数据集成机制为环境中的工具提供统一的 C；控制集成机制为 D 提供支持。界面集成机制使得环境中的所有工具具有 E。

可选答案：

- A: ① CAD                      ② CAI                      ③ CAM                      ④ CASE
- B: ① 开放性和可剪裁性                      ② 开放性和不可剪裁性  
③ 封闭性和可剪裁性                      ④ 封闭性和不可剪裁性
- C: ① 数据类型              ② 数据模式              ③ 数据接口规范              ④ 数据仓库
- D: ① 使各工具具有统一的控制结构  
② 各工具之间的通信、切换、调度和协同工作  
③ 使各工具具有统一的控制接口  
④ 各工具之间的同步开发
- B: ① 同一个界面                      ② 相同的图标和图标的含义  
③ 统一的界面风格和操作方式                      ④ 相同个数的窗口和菜单项

### 思考练习题答案

1. A: ②    B: ⑦    C: ⑤
2. A: ①    B: ①    C: ③    D: ⑥
3. A: ⑦    B: ⑧    C: ⑥    D: ③    E: ②    F: ⑤    G: ④。
4. A: ②    B: ①    C: ⑤    D: ③    E: ⑤
5. A: ⑦    B: ②    C: ⑥    D: ③    E: ②
6. A: ①    B: ③    C: ④    D: ①    E: ①
7. A: ③    B: ②    C: ①    D: ①    E: ②    F: ①    G: ①    H: ②
8. A: ③    B: ⑤    C: ⑥    D: ①    E: ⑤
9. A: ⑤    B: ⑦    C: ③    D: ②    E: ①
10. A: ②    B: ⑤    C: ③    D: ③    E: ②
11. A: ②    B: ①    C: ④    D: ②    E: ①
12. A: ④    B: ②    C: ①    D: ③    E: ②
13. A: ⑤    B: ②    C: ③    D: ①    E: ③
14. A: ②    B: ③    C: ①    D: ①    E: ④
15. A: ②    B: ④    C: ①    D: ③    E: ①



16. A: ④ B: ③ C: ① D: ② E: ③
17. A: ① B: ③ C: ⑤ D: ⑥ E: ⑦
18. A: ② B: ③ C: ③ D: ② E: ④
19. A: ② B: ⑩ C: ⑨ D: ⑧ E: ④
20. A: ③ B: ④ C: ② D: ② E: ② F: ① G: ④  
H: ⑤ I: ① J: ①
21. A: ③ B: ⑥ C: ① D: ② E: ⑥ F: ② G: ① H: ③  
I: ⑤
22. A: ① B: ⑤ C: ① D: ② E: ③
23. 正确叙述有: ④、⑤、⑥、⑦、⑩
24. A: ④ B: ① C: ① D: ③ E: ②
25. A: ① B: ⑤ C: ③ D: ② E: ③
26. A: ② B: ③ C: ① D: ④ E: ③
27. A: ② B: ③ C: ② D: ④ E: ⑤
28. A: ④ B: ③ C: ④ D: ③ E: ② F: ② G: ① H: ③
29. A: ③ B: ① C: ④ D: ② E: ②
30. 不准确的叙述有②、③、④、⑥、⑦
31. A: ④ B: ③ C: ② D: ③ E: ①。
32. 正确的叙述有 ①、③、④、⑥、⑦。
33. A: ③ B: ② C: ⑦ D: ① E: ③ F: ②
34. A: ② B: ① C: ③ D: ④ E: ③
35. A: ④ B: ② C: ③ D: ③ E: ①
36. A: ② B: ③ C: ① D: ② E: ① F: ③ G: ④ H: ⑥  
I: ② J: ④
37. A: ② B: ① C: ③ D: ⑥ E: ③ F: ⑤
38. A: ② B: ④ C: ⑥ D: ② E: ③ F: ⑤ G: ④ H: ①  
I: ④
39. A: ② B: ④ C: ③ D: ② E: ①
40. A: ③ B: ① C: ② D: ① E: ④
41. A: ⑤ B: ③ C: ⑥ D: ② E: ⑤
42. 正确的叙述为 ②、③、⑥、⑧、⑩
43. A: ③ B: ① C: ③ D: ⑤ E: ⑥
44. A: ④ B: ① C: ③ D: ② E: ③

## 第8章 数据结构

### 8.1 内容提要

数据 (data) 是信息的载体, 是描述客观事物属性的数、字符, 以及所有能输入到计算机中并被计算机程序识别和处理的符号的集合。数据结构是指数据对象及其相互关系和构造方法。一个数据结构可用一个二元组表示  $\text{Data\_Structure} = \{D, R\}$ ,  $D$  是某一数据对象, 是数据结构中数据 (称为结点) 的非空有限集合,  $R$  是该对象中所有数据成员之间的关系有限集合。

数据结构中结点与结点之间的关系称为数据的逻辑结构。按数据的逻辑关系, 数据结构可分为线性结构和非线性结构, 其中非线性结构又可分为树形结构和图结构。

数据结构在计算机中的存储形式称为数据的存储结构。典型的存储方式有 4 种, 包括顺序存储结构、链式存储结构、索引存储结构、散列存储结构。

算法与数据结构之间有密切的关系, 算法建立在数据结构的基础上, 选择合理的数据结构可有效地改进算法的效率。算法是过程性的, 按输入-计算-输出的模式解决问题。对算法的分析主要着眼于它的时间代价和空间代价。

#### 8.1.1 线性表

##### 内容要点

通常, 定义线性表为  $n$  个数据元素 (或称为表元) 的有限序列。记为  $L = (a_0, a_1, \dots, a_{n-1})$ 。其中,  $L$  是表名,  $a_i$  是表中的结点, 是不可再分割的数据。 $n$  是表中表元的个数, 也称为表的长度, 若  $n = 0$  叫做空表。线性表是一个有限序列, 意味着表中各个表元是相继排列的, 且每两个相邻表元之间都有直接前驱和直接后继的关系。

线性表主要的存储结构有两种结构: 顺序存储结构和链接存储结构。前者称为顺序表, 后者称为线性链表。顺序表是利用一个连续的存储空间相继地存放线性表的结点; 线性链表是动态分配链表结点, 存放线性表中结点的值, 通过链接指针, 将链接存储结构中各个结点按线性表中的逻辑顺序链接起来。在线性链表中用于存放结点的存储单元可连续, 可不连续。

线性表上主要操作的实现取决于采用哪一种存储结构。存储结构不同, 实现的算法也不同。主要的操作有:

- 表的初始化运算: 将线性表置为空表;
- 求表长度运算: 统计线性表中表元个数;

- 查找运算：查找线性表中第  $i$  个表元位置或查找表中具有给定关键字的表元；
- 插入运算：将新表元插入到线性表第  $i$  个位置上，或插入到具有给定关键字的表元的前面或后面；
- 删除运算：删除线性表第  $i$  个表元或具有给定关键字的表元；
- 读取运算：读取线性表第  $i$  个表元的值；
- 复制运算：复制线性表所有表元到另一个线性表中。

### 学习难点

- 线性表中表元之间的邻接关系实际上反映了表元之间的逻辑关系。在顺序存储中的邻接关系反映在存储位置上是邻接的，但在链接存储中需通过链接指针反映邻接关系，各表元在存储位置上可任意。
- 线性表中各表元是不可再分的。如果表元是表或其他结构，则不再是线性表。
- 在线性表的顺序存储（顺序表）中用一维数组来描述一个连续的存储空间。但一维数组与顺序表不同。一维数组与顺序表都可按结点下标直接（或称随机）存取结点的值，在这一点是相同的，但一维数组中各非空结点可以不相继存放，而顺序表是相继存放的；一维数组只有按数组元素的下标存取该书族元素的操作，而顺序表可以有线性表的所有操作；顺序表的长度是可变的，一维数组的长度一经存储分配是不变的。
- 线性表的链接存储（线性链表）只能通过链接指针顺序访问，这限制了表的查找效率，但给表元的插入和删除带来了方便，因为不需要大量移动表元以腾出插入位置或填补被删表元的位置。
- 通常在定义顺序表时用到一个一维数组，同时需要显式定义数组的最大尺寸  $N$  和顺序表当前的长度  $n$ ：

```
#define N 100
typedef int DataType;
typedef struct {
    DataType data;
} SeqListNode;
SeqListNode a[N], n;
```

在定义线性链表时必须定义链表结点：

```
typedef int DataType;
typedef struct node {
    DataType data;
    struct node *link;
} LinkNode;
```

使用一个指向链表结点的指针标识链表的表头（即首结点）。

LinkedNode \* hpt;

- 线性链表通过指向链表首结点的表头指针进行操作，链表尾结点的链接指针域为空，表示链表收尾。
- 在顺序表的插入操作中需要注意的是表满的判断或插入位置的合理性。如果插入前表已满或指定的插入位置超出表的范围（追加情况例外），则插入失败。
- 在顺序表的删除操作中需要判断表是否空或删除位置是否合理。如果在空表中删除或指定的删除位置超出表的范围，则删除失败。
- 在线性链表中插入有4种方式：在某个指针 p 所指表中的结点之后插入；在链表的首结点之前插入；在链表的链尾插入；在有序链表中插入。需要注意的是，在链表的首结点之前插入时链表的首指针必须改变。在表尾插入与在表中插入处理相同。在有序链表中插入时必须循链查找插入位置，根据是在链表首结点前插入还是在表中或表尾插入，分别进行处理。
- 在线性链表中删除，需要考虑3种情况：当空表时不需删除；当表非空且被删结点恰为表的首结点时，必须改变链表的首指针，然后再执行删除；当删除其他链表结点时，必须先将被删结点从链中摘下，将链中其他结点链接好再执行删除。
- 为了简化链表的插入和删除操作，并统一空表和非空表的操作，通常可在链表的链头设置表头结点。这样，无论是在链表的首结点之前插入，还是在链表的中间或尾部插入，都能够统一处理。同样地，即使删除链表的首结点也不需修改表头指针。

### 8.1.2 栈

#### 内容要点

栈是限定存取只能在表的同一端执行的线性表。允许插入和删除的一端为栈顶，不允许插入和删除的一端为栈底。

栈的逻辑特点是先进后出（FILO）或后进先出（LIFO）。

常见的栈的操作有：

- `IniStack(s)`，栈的初始化操作，使栈 s 成为空栈。
- `Push(s, x)`，将新元素 x 插入到栈 s 的栈顶。
- `Pop(s)`，将栈 s 的栈顶元素从栈中退出。
- `Top(s)`，读取栈 s 的栈顶元素，由函数返回，不改变栈的结构。
- `Empty(s)`，判断栈 s 空否，空则函数返回 1，否则返回 0。

栈的存储表示有两种：顺序存储表示和链接存储表示。前者称为顺序栈，后者称为链式栈。顺序栈类似于顺序表，只是将表的长度 n 换成栈顶指针 top，栈的插入与删除



限定在 top 所指位置进行；链式栈类似于线性链表，栈顶在链表的首结点位置。栈的插入与删除限定在链的首结点位置进行。

顺序栈的定义为：

```
#define MAXN 100
typedef char DataType;
typedef struct {
    DataType data[MAXN];
    int top;
} SeqStack;
```

链式栈的定义为：

```
typedef int DataType;
typedef struct node {
    DataType data;
    struct node *link;
} LinkNode;
typedef LinkNode * LinkStack;
```

栈的用途很多。典型的事例如表达式计算、递归的实现。在表达式计算中，用运算符栈和运算分量栈辅助求值。在递归的实现中，常利用栈记忆各层递归时分配的局部变量、形式参数和返回上一层递归调用的地址。

学习难点

- 栈操作 Pop(s) 与 Top(s) 是有区别的。前者退出栈顶的元素，因此每执行一次 Pop(s) 操作，栈中的元素个数就少一个；后者仅复制出一份栈顶元素，栈中元素个数不发生变化。
- 顺序栈的操作一般在栈顶指针处进行。在顺序栈的实现中，初始化函数用于为栈赋初值，令  $top = -1$ ，置栈为空。
- 在实现进栈操作时，应先判断栈是否已满。如果栈顶指针  $top = N-1$  ( $N$  是栈的容量)，则说明栈中所有位置均已使用，栈已满。这时若再有新元素进栈，将发生栈溢出，程序转入出错处理。如果  $top < N-1$ ，则先让栈顶指针进 1，指到当前可加入新元素的位置，再按栈顶指针所指位置将新元素插入。这个新插入的元素将成为新的栈顶元素。
- 如果在退栈时发现是空栈，即  $top = -1$ ，则退栈操作将执行栈空处理。栈空处理一般不是出错处理，而是使用这个栈的算法结束时需要执行的处理。
- 顺序栈的缺点在于当栈满时要发生溢出，为了避免这种情况，需要为栈设立一个足够大的空间。但如果空间设置得过大，而栈中实际只有几个元素，也是一种空间浪费。此外，当程序中需要多个栈时，因为各个栈所需的空间在运行中

是动态变化着的。如果给几个栈分配同样大小的空间, 可能在实际运行时, 有的栈膨胀得快, 很快就产生了溢出, 而其他的栈可能此时还有许多空闲的空间。这时就必须调整栈的空间, 防止栈的溢出。

- 采用链式栈便于结点的插入与删除, 不存在栈满的问题。在程序中同时使用多个栈的情况下, 用链接表示不仅能够提高效率, 还可以达到共享存储空间的目的。
- 链式栈的栈顶在链表的表头。因此, 新结点的插入和栈顶结点的删除都在链表的表头, 即栈顶进行。
- 因为所有的操作都限定在链表的表头进行, 不需要设定链表的表尾指针。
- 在计算机中执行算术表达式的计算是通过栈来实现的。为了正确执行表达式的计算, 必须明确各个操作符的执行顺序。为此, 为每一个操作符都规定了一个优先级。一个表达式中相邻的两个操作符的计算次序为: 优先级高的先计算; 如果优先级相同, 则自左向右计算; 当使用括号时, 从最内层的括号开始计算。
- 因为直接对算术表达式求值时必须使用两个栈, 分别对运算符和运算分量进行处理, 若设当前扫描到的运算符  $ch$  的优先级为  $icp(ch)$ , 该运算符进栈后的优先级为  $isp(ch)$ , 则可规定各个算术运算符的优先级如下表所示。

运算符	;	(	↑	*/,%	+,-	)
isp	0	1	7	5	3	8
icp	0	8	6	4	2	1

$isp$  也叫做栈内优先数 (in stack priority),  $icp$  叫做栈外优先数 (in coming priority)。下面给出对表达式求值的一般规则:

当读取到的字符  $ch$  为运算分量时进运算分量栈, 再读取下一字符。当读取的字符  $ch$  是运算符时, 比较运算符  $ch$  的优先级  $icp(ch)$  和运算符栈的栈顶元素的优先级  $isp(stack)$ 。若  $icp(ch) > isp(stack)$  时, 则  $ch$  进运算符栈, 再读取下一字符。当  $icp(ch) < isp(stack)$  时, 则从运算符栈退出一个运算符, 从运算分量栈连续退出两个运算分量, 形成运算指令, 其运算结果进运算分量栈。运算符优先数相等  $icp(ch) = isp(stack)$  的情况只出现在括号配对 “)” 或运算符栈的栈底的 “;” 号与读取到的最后的 “;” 号配对时。前者将 “(” 退栈以对消括号, 后者将结束算法。

- 当求解一个问题时, 是通过求解与它具有同样解法的子问题而得到的, 这就是递归。
- 一个递归的求解问题必然包含有终止递归的条件, 当满足一定条件时就终止向下递归, 从而使问题得以解决。
- 解决递归问题的算法称为递归算法, 在递归算法中需要根据递归条件直接或间

接地调用算法本身，当满足终止条件时结束递归调用。

- 对于一些简单的递归问题，如单项递归或尾递归，很容易把它转换为循环问题来解决，从而使编写出的算法更为有效。
- 在计算机中执行递归函数是通过递归工作栈来实现的。每递归调用一层栈中就增加一个记录，包含为该层递归调用新分配的参数、局部变量和调用返回时的返回地址。对于引用参数只保存传送来的实参的地址，可按此地址访问实参；对于传值参数保存实参的副本，在函数中使用该副本进行操作。
- 每层递归调用结束时，都按照本层记忆的返回地址返回到上层的指定位置执行，并且自动作一次退栈操作，使得上一层所使用的参数成为新的栈顶，继续被使用。
- 栈和递归是密切相关的，当编写递归算法时，虽然表面上没有使用栈，但系统执行时会自动建立和使用递归工作栈。

### 8.1.3 队列

#### 内容要点

队列是另一种限定存取位置的线性表。它只允许在表的一端插入，在另一端删除。允许插入的一端叫做队尾（tail），允许删除的一端叫做队首（head）。

队列的特点是先进先出（FIFO）。最先加入队列的元素最先退出队列。

队列的主要操作有：

- `IniQueue (Q)`，队列的初始化操作，使队列 `Q` 成为空队列。
- `EnQueue (Q, x)`，将新元素 `x` 插入到队列 `Q` 的队尾，使之成为新的队尾。
- `DeQueue (Q)`，将队列 `Q` 的队首元素从队列中退出。
- `Front (Q)`，读取队列 `Q` 的队首元素，由函数返回，不改变队列的结构。
- `Empty (Q)`，判断队列 `Q` 空否，空则函数返回 1，否则返回 0。

队列的存储表示有两种：顺序存储表示和链接存储表示。前者称为顺序队列，后者称为链式队列。顺序队列类似于顺序表，只是将表的长度 `n` 换成队列的队首指针 `head` 和队尾指针 `tail`。队列的插入限定在 `tail` 所指位置进行，队列的删除限定在 `head` 所指位置进行；链式队列类似于线性链表，队列的队首位于链表的首结点位置，由指针 `head` 指示，队列的队尾位于链表的最后一个结点位置，用指针 `tail` 指示。新元素插入到 `tail` 所指结点的后面，并将 `tail` 指到该结点；队列的删除限定在 `head` 所指结点位置进行。

顺序队列的定义为：

```
#define MAXN 100
typedef char DataType;
typedef struct {
    DataType data[MAXN];
```

```
int head, tail;  
} SeqQueue;
```

链式栈的定义为:

```
typedef int DataType;  
typedef struct node {  
    DataType data;  
    struct node *link;  
} LinkNode;  
typedef struct {  
    LinkNode * head, * tail;  
} LinkQueue;
```

顺序队列的存储空间一经分配就不能扩充,因此存在队满溢出问题。此外,队首指针与队尾指针向同一方向移动,有可能产生“假溢出”现象:队尾指针超出存储空间报告溢出,但队首指针因为有删除并未停留存储空间最前端,有部分空间没有使用。

为此,顺序队列采取让队首指针和队尾指针都循顺时针方向循环移动的办法,当指针位于存储空间的最后时,一旦需要移动,就自动移到存储空间的最前端。称这种队列为循环队列。

链式队列不存在队列满的问题,只要内存中的空闲空间还能满足动态分配结点的要求,就可以继续向队列中加入新结点。但队列在退出队列元素时需要判断队列空否,队列空则退出队列失败,这称为“下溢”,它不是出错情况,而是表明某种处理的结束。

#### 学习难点

- 队列操作 DeQueue (Q)与 Head (Q)是有区别的。前者退出队列的队首元素,因此每执行一次 DeQueue (Q)操作,队列中的元素个数就少一个;后者仅复制出一份队首元素,栈中元素个数不发生变化。
- 在顺序队列中进行插入和删除时,不需要比较和移动任何元素,只需要修改队尾和队首指针,并向队尾插入元素或从队首取出元素。
- 顺序队列的操作按循环队列组织。初始化时令  $head = tail = 0$ ,置队列为空。按此处理,判队列是否空的条件为  $head == tail$ 。每次向队列加入新元素时,先按队尾指针  $tail$ ,将新元素插入,然后队尾指针进到下一个位置。每次从队首退出一个元素时,先将队首指针  $head$  所指元素保存,再让队首指针进到下一个位置,最后返回保存的原队首元素的值。
- 按循环队列组织队列时,将  $data[MAXN-1]$ 与  $data[0]$ 衔接起来,利用取模运算“%”实现  $head$  与  $tail$  的进一操作:  $head = (head+1) \% MAXN$ ,  $tail = (tail+1) \% MAXN$ 。每当指针进到  $MAXN-1$  时,  $(MAXN-1+1) \% MAXN \rightarrow 0$ 。



- 在顺序队列中, 队首指针 `head` 指向实际队首位置, 队尾指针 `tail` 指向实际队尾的下一位置, 因此, 队列满的条件为  $(tail+1) \% MAXN == head$ 。此时牺牲了一个存储空间, 以区分队列空和队列满的条件。
- 在顺序队列的进队列算法中, 需先判断队列是否已满: 如果队列已满, 执行出错处理, 否则执行队列的插入操作。在顺序队列的出队列算法中, 需先判断队列是否为空: 如果队列为空, 执行队空处理, 否则执行队列的删除操作。
- 在链式队列中, 队列空的条件是 `head == NULL`, 此时是空链表, 至于 `tail` 指针是否为 `NUL`, 可以不考虑。如果有 `head == tail`, 不一定是空队列, 因为 `head` 指针指示队列的实际队首位置, `tail` 指针指示队列的实际队尾位置, 如果链表中只有一个结点, 就有 `head == tail`, 不是空队列, 若有 `head == tail`, 还需有 `head == NULL` 才是空队列。
- 在链式队列中没有附加表头结点, 队首指针 `head` 指示链首结点, 队尾指针 `tail` 指示链尾结点。每次向队列中加入新结点时, 总是将新结点链入在 `tail` 所指结点后面, 再让 `tail` 指针指到新结点, 使之成为新的队尾; 每次从队首退出一个结点时, 总是先判断队列是否为空: 若队列为空, 执行队空处理, 否则保存队首结点的地址, 将队首指针移到下一结点, 再删除原队首结点, 返回该结点中元素的值。
- 队列在计算机中用处较多, 经常用在调度算法中。对于分层结构或图结构, 可用队列实现逐层处理, 在处理上一层的元素时, 利用元素间的关系将下一层元素预先进到队列中, 上一层元素处理完再从队列中顺序退出已保存的下一层的元素进行处理。

### 8.1.4 数组

#### 1. 数组

##### 内容要点

通常, 在程序设计语言中将数组看做是存储于一个连续存储空间中的相同数据类型的数据元素的集合。通过数组元素的下标 (位置序号), 可以找到存放该数组元素的存储地址, 从而可以访问该数组元素的值。

数组有静态数组和动态数组之分。静态数组在说明它时就定义了它的长度并在编译时就分配了它的存储空间, 在整个程序运行期间, 数组长度不会改变, 可通过数组元素的下标直接访问该数组元素的值。如

```
#define MAXN 100
int A[MAXN];           //数组元素的下标范围为 0~MAXN-1
```

动态数组在说明它时只定义了一个指针, 在程序运行时通过动态存储分配命令

malloc()分配它的存储空间,程序运行结束时通过动态释放命令 free()回收已分配的存储空间。如

```
#define MAXN 100
int * A; //定义数组指针
A = (int *) malloc(MAXN * sizeof(int)); //分配数组空间
```

对于动态数组,可以用指针访问数组元素。若设 A 是一个数组头指针,它存放数组第一个元素的地址,用\*(A+i)可以读取第 i 个数组元素的值,用 A++或 A--可以遍历该数组的下一个元素或前一个元素。此外,还可以用 A[i]访问第 i 个数组元素的值。

按数组下标的个数,可把数组分为一维、二维、三维等。一维数组中的每个元素只包含有一个下标,二维数组中的每个元素包含有两个下标,第一个称为行下标,第二个称为列下标。同样,三维数组包含有三个下标,每个元素的位置由一组三个下标值惟一确定。

数组的存储结构是顺序结构,即数组中第 i+1 个元素紧接着存储在第 i 个元素的存储位置的后面。如对于一维数组 a[n],则每个元素 a[i]的存储位置的首字节地址为:

$$\text{Address } a[i] = a + i * \text{len} \quad (0 \leq i \leq n-1)$$

其中,数组名 a 同时为表示该数组首地址的符号常量, len 表示数组 a 中元素类型的长度(即每个元素所占用的字节数)。由上述公式可知:元素 a[0]的存储地址为 a, a[1]的存储地址为 a+1\*len, a[2]的存储地址为 a+2\*len, ..., a[n-1]的存储地址为 a+(n-1)\*len。

对于二维数组 a[m][n],为能根据它的数组元素的下标计算出在相应一维数组中对应的下标,需要区分两种存储方式,即行优先顺序和列优先顺序。按照行优先的顺序,所有数组元素按行向量依次排列,设二维数组 a[m][n] 第一个元素 a[0][0] 存放于相应一维数组中的第 0 号位置,每个元素占 1 个元素的空间,那么,任一数组元素 a[i][j] 在相应一维数组中的存放地址利用递推公式计算得 i\*n+j。

长整数的运算可以利用数组进行。设有一个长整数 m,它可以表示为

$$m = a[1] * 10^0 + a[2] * 10^1 + a[3] * 10^2 + \dots + a[k-1] * 10^{k-2} + a[k] * 10^{k-1}$$

利用一个整数数组 a[k+1],存放各十进位的数值, a[i]存放  $10^{i-1}$  的数值 ( $0 \leq a[i] \leq 9$ ),在 a[0]中存放 k。

```
#define MAXN 1000 //数组最大长度
int a[MAXN+1]; //存放长整数的数组, a[0]存储长整数最高位数
```

利用这个存储结构,可实现长整数的输入、输出、相加、相乘等运算。

#### 学习难点

- 二维数组可看做是一维数组的推广或嵌套,即首先把它看做是按行下标顺序排列的一维数组,该数组中的每个元素又都是按列下标顺序排列的一维数组。如对于一个二维数组 b[m][n],可看做为一维数组 b[m],所含元素依次为 b[0],

$b[1], \dots, b[m-1]$ , 其中每一个元素  $b[i]$  ( $0 \leq i \leq m-1$ ) 又都是一个含有  $n$  个元素的一维数组, 所含元素依次为  $b[i][0], b[i][1], \dots, b[i][n-1]$ 。

- 二维数组是最简单的非线性结构, 它的每一个数组元素有 0 个或 2 个直接前驱, 有 0 个或 2 个直接后继。
- 对于一个  $m$  行  $n$  列的二维数组  $a[m][n]$ , 如果以行为主序存储 (即按行存储于一个一维数组  $b[m \times n]$  中), 则对于任一数组元素  $a[i][j]$  ( $0 \leq i \leq m-1, 0 \leq j \leq n-1$ ), 有
 
$$\text{Loc}(a[i][j]) = \text{Loc}(a[0][0]) + (i * n + j) * l$$

其中,  $\text{Loc}$  表示存储地址,  $l$  表示每个数组元素的存储单元数。

- 对于一个  $n \times n$  的对称矩阵  $a$ , 为了节约存储, 可以只存对角线及对角线以上的元素, 或者只存对角线或对角线以下的元素。前者称为上三角矩阵, 后者称为下三角矩阵。把它们按行存放于一个一维数组  $b$  中, 称为对称矩阵  $a$  的压缩存储。则数组  $b$  共有  $n*(n+1)/2$  个元素。
- 在只存上三角部分情形, 若  $i \leq j$ , 则数组元素  $a[i][j]$  前面有  $i$  行 ( $0 \sim i-1$ ), 第  $k$  行有  $n-k+1$  个元素。在第  $i$  行中, 从对角线算起, 第  $j$  号元素前面有  $j-i$  个元素, 因此, 数组元素  $a[i][j]$  在数组  $b$  中的存放位置为  $(2n-i+1)*i/2+j-i = (2n-i-1)*i/2+j$ ; 若  $i > j$ , 数组元素  $a[i][j]$  在数组  $b$  中没有存放, 可以找它的对称元素  $a[j][i]$ , 应在  $b$  的  $(2n-j-1)*j/2+i$  位置。
- 在只存下三角部分情形, 若  $i \geq j$ , 则数组元素  $a[i][j]$  前面有  $i$  行 ( $0 \sim i-1$ ), 第  $k$  行有  $k+1$  个元素, 在第  $i$  行中, 第  $j$  号元素前面有  $j$  个元素, 因此, 数组元素  $a[i][j]$  在数组  $b$  中的存放位置为  $i*(i+1)/2+j$ , 当  $i < j$  时, 数组元素  $a[i][j]$  在数组  $b$  中没有存放, 可以找它的对称元素  $a[j][i]$ , 应在  $b$  的  $j*(j+1)/2+i$  位置。
- 在实际应用中经常遇到三对角矩阵, 在该矩阵中除主对角线及在主对角线上下最临近的两条对角线上的元素外, 所有其他元素均为 0。现在要将三对角矩阵  $a$  中三条对角线上的元素按行存放在一维数组  $b$  中, 则总共有  $3n-2$  个非零元素。若  $a_{00}$  存放于  $b[0]$ , 那么  $a$  在三条对角线上的元素  $a_{ij}$  ( $0 \leq i \leq n-1, i-1 \leq j \leq i+1$ ) 在一维数组  $b$  中的存放位置  $2*i+j$ 。
- 对于上述三对角矩阵的压缩存储  $b$ , 若已知某在三条对角线上的元素  $a_{ij}$  ( $0 \leq i \leq n-1, i-1 \leq j \leq i+1$ ) 在一维数组  $b$  中的存放位置  $k$ , 则可求得该元素的行下标为  $i = \lfloor (k+1)/3 \rfloor$ , 列下标为  $j = k - 2*i$ 。

## 2. 稀疏矩阵

### 内容要点

稀疏矩阵是矩阵中的一种特殊情况, 其非零元素的个数远远小于零元素的个数。在实际应用中, 稀疏矩阵一般都比较小, 非零元素所占的比例都比较小。

在计算机中存储矩阵的一般方法是采用二维数组, 其优点是可以随机访问每一个元素, 因而能够较容易地实现矩阵的各种运算, 如转置、加法、乘法等。但对于稀疏矩阵

来说,采用二维数组的存储方法既浪费大量的存储单元用来存放零元素,又要在运算中花费大量的时间来进行零元素的无效计算,显然是不可取的。一种较好的方法是:只考虑存储占元素中极少数的非零元素。

对于稀疏矩阵中的每个非零元素,可用它所在的行号、列号以及元素值这三元组 $(i, j, a_{ij})$ 来表示。若把所有的三元组按行号为主序、列号为辅序(当行号相同时再考虑列号次序)进行排列,则就构成了一个表示稀疏矩阵的三元组表。

稀疏矩阵采用三元组线性表表示后,可以使用顺序或链接方式存储,从而比采用二维数组存储要大大地节省存储空间。

稀疏矩阵的顺序存储就是对其相应的三元组表进行顺序存储。假定每个非零元素的三元组用如下记录结构定义:

```
typedef int DataType;
typedef struct {
    int row, col;           //稀疏矩阵中表示非零元素的三元组
    DataType val;          //非零元素所在的行号、列号
} Triple;                 //非零元素的值
```

一个稀疏矩阵的顺序存储类型定义如下:

```
#define MaxTerms 1000
typedef struct {
    int m, n, t;           //稀疏矩阵定义
    Triple sm[MaxTerms];  //稀疏矩阵的行数、列数和非零元素个数
} SMatrix;               //三元组表
```

稀疏矩阵的十字链表是既带行指针向量又带列指针向量的链接存储。在这种链接存储中,每一个三元组结点按矩阵元素所在的行号 $i$ ,列号 $j$ ,链接入第 $i$ 个行链表和第 $j$ 个列链表中,即处于所在的行链表和列链表的交汇处。

在十字链表中,每个结点的类型可定义为:

```
typedef int DataType;
typedef struct node {
    int row, col;          //非零元素的结点
    DataType val;          //元素所在行号和列号
    struct node * right, * down; //元素值
} CrossNode;             //行链接指针和列链接指针
```

在稀疏矩阵的十字链表表示中,需要使用两个指针向量,一个是行指针向量,用来存储行链表的表首指针(right),另一个是列指针向量,用来存储列链表的表首指针(down)。为充分利用结点空间,第 $i$ 个行链表的表首指针与第 $i$ 个列链表的表首指针合



并在行(列)链表表头结点向量中的第  $i$  个结点中。

稀疏矩阵的十字链表表示定义如下:

```
#define MaxRow&Cols 100
typedef struct CLMatrix {           //稀疏矩阵的十字链表表示
    int m, n, t;                   //矩阵的行数、列数和非零元素个数
    CrossNode* rv[MaxRow&Cols+1]; //各行(列)链表的表头结点向量
};
```

### 学习难点

- 稀疏矩阵的输入: 假定稀疏矩阵采用 SMatrix 类型存储, 按照对应三元组表中三元组排列的次序输入, 每行输入一个三元组, 包括非零元素的行号、列号和元素值, 当输入完所有三元组后, 以输入一个特殊的三元组 0, 0, 0 结束整个输入过程。
- 稀疏矩阵的转置: 第一种方法对稀疏矩阵中的 sm 数组进行  $n$  次扫描 ( $n$  为稀疏矩阵的列数, 即转置矩阵的行数) 才能完成。具体地, 第一次扫描把 col 域的值等于 0 (即列号为 0) 的三元组 (对应转置矩阵中第 0 行非零元素) 按照从上到下的顺序, 行、列号互换, 连同元素值写入到对应转置矩阵的 sm 数组中, 第二次扫描把 col 域的值等于 1 (即列号为 1) 的三元组 (对应转置矩阵中第 1 行非零元素) 按照从上到下的顺序接着写入到对应转置矩阵的 sm 数组中, 依此类推。
- 稀疏矩阵的转置: 第二种方法只需对稀疏矩阵中的 sm 数组进行两次扫描, 第一次扫描统计出稀疏矩阵中每一列 (即对应转置矩阵中每一行) 非零元素的个数, 由此求出每一列的第一个非零元素 (即对应转置矩阵中每一行的第一个非零元素) 在对应转置矩阵的 sm 数组中的位置, 第二次扫描把稀疏矩阵 sm 数组中的每一个三元组行、列号互换, 连同其元素值, 按行号写入到对应转置矩阵的 sm 数组中预定的位置上。
- 稀疏矩阵的加法: 假定采用三元组表进行稀疏矩阵的加法运算, 设  $M_1$  和  $M_2$  为两个相加的矩阵,  $M$  为结果矩阵。两矩阵相加的前提条件是: 两矩阵的大小相同, 即行数和列数分别对应相等。两矩阵相加的结果仍为一个具有相同大小的矩阵。算法要求对  $M_1$  和  $M_2$  的三元组表从上向下顺序检测, 行 / 列号都相等的, 对应元素值相加, 结果若不为零, 则送结果矩阵  $M$  的三元组表, 否则不送结果矩阵  $M$  的三元组表; 行 / 列号不相等的, 行 / 列号小的送结果矩阵  $M$  的三元组表。矩阵  $M$  中每个行单链表仍然要按列号有序, 它是对  $M_1$  和  $M_2$  中对应行单链表的按列号有序的合并结果。当  $M_1$  和  $M_2$  中对应行单链表的两个结点分别具有相同的行号和列号时, 若它们的元素值之和为 0, 则不在结果矩阵中建立结点, 只有当其和不为 0 或者列号不同时, 才需要在结果矩阵中

建立结点。

- 设有两个分别为  $m \times n$  和  $n \times l$  的矩阵  $M1$  和  $M2$ ，它们相乘后的结果矩阵  $M$  为  $m \times l$  的。对于  $M$  中的每一个矩阵元素  $M[i][j]$ ， $0 \leq i \leq m-1$ ， $0 \leq j \leq l-1$ ：

$$M[i][j] = \sum_{k=0}^{n-1} M1[i][k] * M2[k][j]$$

可以设想，在做乘法之前先让所有的  $M[i][j]$  都初始化为 0，然后对于任一非零元素  $M1[i][k]$  ( $0 \leq k \leq n-1$ )，按照公式，寻找所有的行号为  $k$  的  $M2[k][j]$ ，将  $M1[i][k] * M2[k][j]$  相乘的结果累加到  $M[i][j]$  中去。当所有的  $M1[i][k]$  都处理完成， $M[i][j]$  就计算完成了。

### 8.1.5 字符串

#### 内容要点

字符串也简称为串，是  $n$  ( $n \geq 0$ ) 个字符的一个有限序列。通常可记为

$$S = 'a_0 a_1 a_2 \cdots a_{n-1}'$$

其中， $S$  是串名，可以是串变量名，也可以是串常量名。用引号 '...' 或 "..." 作为分界符括起来的叫做串值，其内的  $a_i$  是串中的字符 ( $0 \leq i < n$ )， $n$  是串中的字符个数，也叫做串的长度，它不包括作为分界符的引号，也不包括串结束符 '\0'。长度为零的串叫做空串，除串结束符外，它不包含任何其他字符。

注意要区别空串和空白串，空白串的长度不为零，除串结束符外，它包含的其他字符均为空格。

若一个字符串不为空，从这个串中连续取出若干个字符组成的串叫做原串的子串。例如，如果  $S = 'maintenance'$ ， $P = 'ten'$ ，则  $P$  是  $S$  的子串，它是在  $S$  中从第 4 个字符开始，连续取 3 个字符组成的串。一般称子串的第 0 个字符在串中的位置为子串在串中的位置。如在上例中  $P$  在  $S$  中的位置为 4。

特别地，空串是任意串的子串；任一串是它自身的子串。除它本身以外，一个串的其他子串都是它的真子串。

字符串在解决实际问题时有广泛的应用，因此要考虑可能的串操作，为此，先定义串的存储表示。在 C/C++ 中常使用两种表示定义串的存储结构：

- 定长顺序存储表示

定长顺序存储表示类似于线性表的顺序存储表示，根据预定义的串的最大长度，静态地分配所需要的存储空间  $ch[ ]$ ，在  $ch[0]$  中存放串的当前长度，即实际串中字符的个数。

```
#define MAXLEN 256 // 预定义的串长度
typedef char SString[MAXLEN+1]; // 静态分配的字符串（类型）
```

- 堆分配的存储表示

这种字符串的存储空间不是在类型说明中定义的,而是在程序运行期间动态分配的。在C中使用一种称为“堆”的自由存储区来动态分配串所需要的存储空间,使用动态存储分配命令 `malloc()` 和动态存储释放命令 `free()` 来管理串的存储空间。这种堆分配的存储表示定义为:

```
typedef struct {                                //串的定义
    char *ch;                                   //串存储数组的首指针
    int length;                                 //串长度
} Hstring;
```

### 学习难点

- 串的复制: 从一个字符串常量 `char *init` 复制所有字符到目标串 `T`。应注意的是,在字符串常量的尾部应有一个串结束符 `'\0'`, 可作为复制串中字符的结束控制标志。
- 串的连接: 在目标串 `T` 后面连接另一个串 `S`。应注意的是,需要预先估计两个串的长度相加后是否会超过串 `T` 的最大长度。若超出 `T` 的最大长度,只能连接串 `S` 中部分字符; 否则串 `S` 的所有字符都能连接到串 `T` 后面。
- 取串 `T` 的子串: 在操作中,首先要对提取子串的开始位置和提取的字符个数进行合理性检查。如果要提取子串最后一个字符在串中的位置超出了串的最大允许位置,要做出错处理。
- 求子串在串中的位置: 问题的提法是,设有两个串 `T` 和 `p`, 若打算在串 `T` 中查找是否有与串 `p` 相等的子串,则称串 `T` 为目标, `p` 为模式。并称查找模式在目标中的匹配位置的运算为模式匹配。模式匹配的一种最简单的做法是用 `p` 的字符依次与 `T` 中的字符做比较,如果匹配成功,返回模式第 0 个字符 `p0` 在目标中匹配的位置; 如果在其中某个位置  $i: t_i \neq p_i$ , 比较不等,这时可将模式 `p` 右移一位,用 `p` 中字符从头开始与 `T` 中字符依次比较。

一种高效的模式匹配算法叫做 KMP 算法,这是一种无回溯的算法。如果在做某一趟匹配比较时,模式 `p` 从第 0 号位置比较到第  $i$  号位置“失配”,这时在模式 `p` 中寻找一个  $k$  值,使得  $p_0 p_1 \dots p_k = p_{i-k} p_{i-k+1} \dots p_i$ , 如此,目标 `T` 在下一趟比较时,扫描指针不必回溯,从此处继续向下进行匹配比较,而在模式 `p` 中,扫描指针应退回到 `pk` 位置。

$k$  的确定方法,仅依赖于模式 `p` 本身前  $i$  个字符的构成,与目标无关。因此,称  $k$  是 `pi` 的失败链接值。求模式 `p` 各位的失败链接值的算法如下:

① 设置一个数组 `next[]`, 其元素个数与模式 `p` 中字符个数相等,依次存放模式 `p` 各字符的失败链接值。

② 置 `next[0] = -1`, 然后假定在 `next[0], next[1], ..., next[j-1]` 已求得的情况下,求

next[j]。有三种情况:

- 如果  $\text{next}[j-1] == k$ , 又有  $p[k] == p[j-1]$ , 则置  $\text{next}[j] = \text{next}[j-1] + 1$ 。
- 如果  $p[k] != p[j-1]$ , 令  $k_1 = \text{next}[k]$ , 如果  $p[k_1] == p[j-1]$ , 则置  $\text{next}[j] = k_1 + 1$ 。
- 如果  $p[k_1] != p[j-1]$ , 则按照  $p[k_1]$  的失败链接值继续寻找, 直到找到一个失败链接值  $k_n$ , 使得  $p[k_n] == p[j-1]$  或者  $k_n == -1$ , 这时都置  $\text{next}[j] = k_n + 1$ 。

### 8.1.6 树与二叉树

#### 1. 树

##### 内容要点

树是树形结构的简称, 它是一种重要的非线性结构。树的定义是:

树是由一个或多个结点组成的有限集合, 它有且仅有一个称作根的结点, 其余的结点可分为  $m$  棵 ( $m \geq 0$ ) 互不相交的子树 (称作根的子树), 每棵子树又同样是一棵树。

显然, 树的定义是递归的, 树是一种递归的数据结构。树的递归定义, 将为以后实现树的各种运算提供方便。

当树中某结点有多个子树时, 其子树的顺序一般是任意的, 这种树是无序树。如果对一个度为  $m$  的树规定了各个子树的序号, 这种树是有序树, 子树的序号从左向右用  $1, 2, \dots$  编号。

树的存储结构常用的有三种:

- 双亲表示法: 用一个一维数组存储树中的结点, 同时在每个结点中附设一个指示器指示该结点的双亲结点在数组中的位置。

```
#define MAXSIZE 50                //树中最多结点个数
typedef char DataType;            //结点数据类型
typedef struct node {              //树结点类型定义
    DataType data;                 //结点数据
    int parent;                    //结点双亲
} PTNode;

typedef struct {                   //树类型定义
    PTNode tnode[MAXSIZE];        //双亲表示
    int n;                         //现有结点数
} Ptree;
```

这种双亲表示法利用了树中每个结点最多只有 0 个或 1 个双亲结点的特点。

- 树的标准存储表示: 树的这种表示中每个结点由两部分组成 (结点数据 `data`, 指向子女结点的指针数组 `child[]`)。对于度为  $M$  的树, 结点指针数组中有  $M$  个指针。



```

#define M 10 //树的度
typedef char DataType; //结点数据类型
typedef struct node { //树结点类型定义
    DataType data; //结点的数据
    struct node * child[M]; //结点的子女指针数组
} CTNode;
CTNode * root; //树的根结点指针

```

树的根结点指针是一个指针变量，它指示树的根结点。有的教科书称这种结构为 M 叉链表。另外，可以在结点定义中增加一个指向其双亲结点的指针 `parent`，使得寻找某树结点的子女和双亲都很方便。

- 树的子女兄弟表示：树的每一个结点由三个部分组成（结点数据 `data`，最左子女结点指针 `lch`，右兄弟结点指针 `rsib`）。

```

typedef char DataType; //结点数据类型
typedef struct node { //树结点类型定义
    DataType data; //结点的数据
    struct node * lchild, * rsibling; //结点的最左子女和右兄弟结点指针
} LTNode;
LTNode * root; //树的根结点指针

```

这种存储表示法比较灵活。结点的左链是子女链，沿此链可找到结点的第一个子女；结点的右链是兄弟链，沿此链可找到结点的所有兄弟结点。

树的遍历是指按某种次序访问树中的所有结点，使得每个结点访问一次且仅访问一次。

树的遍历包括按前序方向遍历、按后序方向遍历、按层次遍历、遍访所有叶结点。

- 按前序方向遍历定义为：先访问根结点，然后从左到右依次先根遍历每棵子树，此遍历过程是一个递归过程。
- 按后序方向遍历定义为：先从左到右依次先根遍历每棵子树，然后访问根结点。此遍历过程也是一个递归过程。
- 按层次方向遍历定义为：先访问第一层结点（即树的根结点），再从左到右访问第二层结点，依次按层访问，直到访问完最深一层结点为止。此遍历过程不是一个递归过程。
- 按自左向右的顺序访问树中所有的叶结点。

#### 学习难点

- 树的定义在各教科书上是不同的，在本教材中树的定义中至少必须有一个结点，不能是空树。故在考试时要注意题中对树的定义是哪一种。
- 特别需要注意的是，树中根结点所在的层次为 0。某些数据结构教科书定义根

结点所在层次为 1, 在考试时也需注意问题的提法。因为由此带来的问题是计算树的深度或高度 (即离根最远的叶结点所在层次) 的结果可能差 1。

- 树中除根结点没有直接前驱 (双亲) 外, 其他各结点只有一个直接前驱 (双亲)。但各结点可以有零个或多个子女结点。因此, 在有  $n$  个结点的树中有  $n-1$  条边。
- 树的第一种存储表示是双亲表示。在这种表示中, 求一个结点的双亲的操作一步到位, 求子女或兄弟的操作可能要检测树中的所有结点。
- 树的标准存储表示可能浪费较多的指针。如果一棵树中有  $n$  个结点, 有  $n*M$  个指针, 其中有  $n-1$  个非空指针, 有  $n*M - (n-1) = n*(M-1) + 1$  个空指针。
- 树的子女兄弟存储表示只有  $n+1$  个空指针, 存储浪费少。如果需要寻找一个结点的所有子女, 只需先从此结点沿左链找到它的最左子女结点, 在循这个子女结点的右链可找到它的所有兄弟。
- 树的前序遍历也叫做深度优先遍历, 按层次遍历也叫做广度优先遍历。
- 按自左向右的顺序访问树中结点时可以用前几种遍历方式实现。

## 2. 二叉树

### 内容要点

二叉树的递归定义为: 二叉树或者是一棵空树, 或者是一棵由一个根结点和两棵互不相交的分别称作根的左子树和右子树所组成的非空树, 左子树和右子树又同样是一棵二叉树。这是一个递归的定义。

另外还有几个特殊的二叉树, 即满二叉树、完全二叉树和丰满树。

设二叉树  $T$  有  $n$  个结点, 令  $k = \lceil \log_2(n+1) \rceil - 1$ ,  $r = n - (2^k - 1)$ , 则  $k$  代表离根最远的结点所处层次, 即二叉树的深度,  $r$  代表第  $k$  层结点个数。如果其中  $2^k - 1$  个结点放满第 0 到第  $k-1$  层, 则

- (1) 若  $0 < r \leq 2^k$ , 且这  $r$  个结点集中存放在第  $k$  层的左侧, 则称  $T$  是一棵完全二叉树;
- (2) 特别地, 若  $r = 2^k$ , 则称  $T$  是一棵满二叉树, 满二叉树是完全二叉树;
- (3) 若  $0 < r \leq 2^k$ , 但这  $r$  个结点随意分布在第  $k$  层上, 则称  $T$  为丰满树, 或称为理想平衡树。

二叉树具有下列重要性质:

性质 1 二叉树上叶结点的结点数等于度为 2 的结点的结点数加 1。

性质 2 二叉树上第  $i$  层上至多有  $2^i$  个结点 ( $i \geq 0$ )。

性质 3 深度为  $h$  的二叉树至多有  $2^{h+1} - 1$  个结点 ( $h \geq -1$ )。

性质 4 对完全二叉树中编号为  $i$  的结点 ( $1 \leq i \leq n, n \geq 1, n$  为结点数) 有:

(1) 若  $i \leq \lfloor n/2 \rfloor$ , 即  $2i \leq n$ , 则编号为  $i$  的结点为分支结点, 否则为叶结点。

(2) 若  $n$  为奇数, 则每个分支结点都有左子女和右子女; 若  $n$  为偶数, 则编号最大的分支结点 (编号为  $n/2$ ) 只有左子女, 没有右子女, 其余分支结点左右子女都有。

(3) 若编号为  $i$  的结点有左子女, 则左子女结点的编号为  $2i$ ; 若编号为  $i$  的结点有

右子女, 则右子女结点的编号为  $2i+1$ 。

(4) 除树的根结点外, 若一个结点的编号为  $i$ , 则它的双亲结点的编号为  $\lfloor n/2 \rfloor$ , 就是说, 当  $i$  为偶数时, 其双亲结点的编号为  $i/2$ , 它是双亲结点的左子女, 当  $i$  为奇数时, 其双亲结点的编号为  $(i-1)/2$ , 它是双亲结点的右子女。

性质 5 具有  $n$  个 ( $n > 0$ ) 结点的完全二叉树的深度为  $\lceil \log_2(n+1) \rceil - 1$  或  $\lfloor \log_2 n \rfloor$ 。

二叉树的存储结构多用二叉链表, 其定义为

```
typedef char DataType;           // 结点数据类型
typedef struct node {             // 二叉树结点类型定义
    DataType data;                 // 结点的数据
    struct node *lchild, *rchild; // 结点的左、右子树指针
} BTreeNode;
BTreeNode * root;                 // 二叉树根指针
```

二叉树常用的遍历算法有前序、中序、后序和层次序遍历。前序遍历的提法是: 若二叉树不空, 则先访问根结点, 然后前序遍历根的左子树, 前序遍历根的右子树。中序遍历的提法是: 若二叉树不空, 则先中序遍历根的左子树, 然后访问根结点, 再中序遍历根的右子树。后序遍历的提法是: 若二叉树不空, 则后序遍历根的左子树, 后序遍历根的右子树, 最后访问根结点。层次序遍历的提法是: 从根开始, 逐层访问各层的结点。

#### 学习难点

- 二叉树与树不同。二叉树可以是空树, 树不可以是空树; 二叉树结点的子树有左右之分, 即使子树只有一个, 也分左右, 树的结点的子树没有左右之分; 二叉树结点的度有 0、1 和 2, 树的结点没有这个限制。
- 需要明确的是, 二叉树的某一结点没有左子女或右子女, 不表明没有子树, 而是表明有子树, 但子树是空的。
- 如果将一棵有  $n$  个结点的完全二叉树自顶向下, 同一层自左向右连续给结点编号 0, 1, 2, ...,  $n-1$ , 然后按此结点编号将树中各结点顺序地存放于一个一维数组中, 并简称编号为  $i$  的结点为结点  $i$  ( $0 \leq i \leq n-1$ ), 则有以下关系:
  - ① 若  $i=0$ , 则结点  $i$  为根, 无双亲; 若  $i>0$ , 则结点  $i$  的双亲为结点  $\lfloor (i-1)/2 \rfloor$ 。
  - ② 若  $2*i+1 < n$ , 则结点  $i$  的左子女为结点  $2*i+1$ 。
  - ③ 若  $2*i+2 < n$ , 则结点  $i$  的右子女为结点  $2*i+2$ 。
  - ④ 若结点编号  $i$  为偶数, 且  $i \neq 0$ , 则它的左兄弟为结点  $i-1$ 。
  - ⑤ 若结点编号  $i$  为奇数, 且  $i \neq n-1$ , 则它的右兄弟为结点  $i+1$ 。
  - ⑥ 结点  $i$  所在层次为  $\lfloor \log_2(i+1) \rfloor$ 。
- 二叉树的前序、中序、后序遍历算法是递归的算法。它们在二叉树中走过的路线是相同的。如果利用栈, 可以写出相应的非递归算法。但层次序遍历不是递归算法, 需要利用队列来实现逐层访问。



- 完全二叉树用一维数组存储非常有效, 但用一维数组表示一般二叉树不很理想。此外, 在一棵树中进行插入和删除时, 为了反映结点层次的变动, 可能需要移动许多结点, 降低了算法效率。而使用链接表示, 可以克服这些缺点。

### 3. 二叉查找树

#### 内容要点

二叉查找树又称二叉排序树, 它或是一棵空树, 或是一棵具有如下特性的非空二叉树:

- (1) 若它的左子树非空, 则左子树上所有结点的关键字均小于根结点的关键字;
- (2) 若它的右子树非空, 则右子树上所有结点的关键字均大于(若允许具有相同的关键字的结点存在, 则大于等于)根结点的关键字;
- (3) 左、右子树本身又各是一棵二叉查找树。

在二叉查找树上进行查找, 是一个从根结点开始, 沿某一个分支逐层向下进行比较判等的过程。假设想要在二叉查找树中查找关键字为  $x$  的元素, 查找过程从根结点开始。如果根指针为 NULL, 则查找不成功; 否则用给定值  $x$  与根结点的关键字进行比较: 如果给定值等于根结点的关键字, 则查找成功, 返回查找成功信息, 并报告查找到的结点地址。如果给定值小于根结点的关键字, 则继续递归查找根结点的左子树; 否则递归查找根结点的右子树。

为了向二叉查找树中插入一个新元素, 必须先检查这个元素是否在树中已经存在。如果使用查找算法在树中检查要插入元素时查找成功, 说明树中已经有这个元素, 不再插入; 如果查找不成功, 说明树中原来没有关键字等于给定值的结点, 把新元素加到查找操作停止的地方。

在二叉查找树中删除一个结点时, 需要分情况讨论。如果想要删除叶结点, 只需将其双亲结点指向它的指针清零, 再释放它即可。如果被删结点缺右子树, 可以拿它的左子女结点顶替它的位置, 再释放它。如果被删结点缺左子树, 可以拿它的右子女结点顶替它的位置, 再释放它。这些都比较简单。如果被删结点左、右子树都存在, 可以在它的右子树中寻找中序下的第一个结点(关键字最小), 用它的值填补到被删结点中, 再来处理这个结点的删除问题, 这是一个递归处理。当然, 也可以在被删结点的左子树中找中序下的最后一个结点(关键字最大), 用它来填补被删结点。

如果插入结点的关键字序列是一个按值递增的有序序列, 会导致建立一个右斜的单支树, 使得查找性能显著变坏, 其平均比较次数达到  $n$  的数量级(用  $O(n)$  表示)。如果建立起来的二叉查找树的左右子树的深度之差的绝对值不超过 1, 这样的二叉查找树称为平衡二叉树, 简称为平衡树。它的查找效率达到  $O(\log_2 n)$ 。

#### 学习难点

- 在二叉查找树上进行查找可以是一个递归的过程, 也可以用非递归过程实现。这个非递归查找过程需要一个查找指针  $*p$  和它的双亲指针  $*pr$ , 初始时  $p = root$ ,



$pr = \text{NULL}$ , 如果  $p$  所指结点的关键字等于给定值, 则查找成功,  $p$  返回找到结点的地址,  $pr$  返回它的双亲结点地址; 若比较不等,  $pr$  记忆  $p$  所指结点地址, 再用  $p$  所指结点的关键字与给定值做比较, 根据比较结果, 指向下一层某一子女结点, 如果指针  $p$  在逐层下落过程中总是比较不等, 最后等于  $\text{NULL}$ , 查找失败,  $pr$  返回给定值应插入的位置。这个非递归的循环过程可以不用栈, 关键字比较次数等于树的深度加 1。

- 在向二叉查找树中插入新结点时, 新结点必须作为叶结点插入。每次寻找插入位置时必须从根开始, 利用查找算法来处理。
- 在二叉查找树中删除一个结点时, 必须将因删除结点而断开的二叉链表重新链接起来, 同时确保二叉查找树的性质不会失去。此外, 为了保证在执行删除后, 树的查找性能不至于降低, 还需要防止重新链接后树的高度不能增加。在删除算法中所有这些因素都应当体现。

#### 4. 穿线树

##### 内容要点

在二叉树结点的空指针域中存放该结点在某次遍历次序下的前驱结点或后继结点的指针叫做线索, 对一棵二叉树中的所有结点的空指针域按照某种遍历次序加线索的过程叫做线索化, 被线索化了的二叉树称作穿线树。

在空的左指针域中存放的指向其前驱结点的指针叫做前驱线索, 在空的右指针域中存放的指向其后继结点的指针叫做后继线索。

对一棵二叉树进行某种遍历次序的线索化, 就是对该二叉树进行这种遍历的过程, 只是将访问根结点的操作改为对指针域为空的结点加线索。具体做法是:

(1) 若前驱结点不为空, 同时前驱结点的右指针域为空, 则将根结点的地址赋给前驱结点的右指针域, 将前驱结点的右线索标志置 1;

(2) 若根结点的左指针域为空, 则把前驱结点的地址赋给根结点的左指针域, 同时将根结点的左线索标志域置 1;

(3) 将根结点地址赋给保存前驱结点指针的变量, 以便访问下一个结点时, 此根结点成为前驱结点。

利用线索遍历中序穿线树时, 可以先利用  $\text{First}()$  运算找到穿线树在中序下的第一个结点, 然后利用求后继结点的运算  $\text{Next}()$  按中序次序逐个访问, 直到穿线树的最后一个结点。在遍历过程中可以不用栈, 但是在穿线树中不是所有结点都能直接找到其后继的: 如果在结点的右指针域中存放的是后继线索, 可以直接按后继线索找到该结点的直接后继; 否则右指针域中是右子女指针, 需要通过一定运算才能找到它的后继。

在中序穿线树中某指定结点  $*p$  之前插入新结点  $*q$ , 使之成为结点  $*p$  的直接前驱, 可以首先找到结点  $*p$  的中序下的直接前驱  $*t$ , 将  $*q$  插入到  $*t$  右子女位置, 修改结点  $*p$  的直接前驱信息 (可能的话)、结点  $*t$  的右子女信息以及结点  $*q$  的前驱和后继线索。

但如果结点\*p没有直接前驱,则新结点\*q成为结点\*p的左子女结点,修改结点\*p的左子女信息和结点\*q的前驱和后继线索。

在中序穿线树中某指定结点\*p之后插入新结点\*q,使之成为结点\*p的直接后继的情况与插入直接前驱的情况是对称的,左改右,右改左,前驱改后继即可。

#### 学习难点

- 穿线树直接与树的存储结构有关,它在树的二叉链表中加入了线索标志,用以区分线索和子女指针。
- 在中序穿线树中的 First() 运算的实现是从根结点开始,沿结点左链一直走到底,直到结点的左线索标志为 1 为止。最左下的结点(不一定是叶结点)就是该穿线树中序下第一个结点。
- 在中序穿线树中找寻某结点\*p的后继的运算 Next() 的实现也比较简单:首先判断结点\*p的右线索标志是否为 1,若为 1 则右指针域存放的是该结点的后继线索,从中可得到后继结点地址;若不为 1,在结点右指针域存放的是右子女指针,从它的右子女结点出发,找寻该子树在中序下的第一个结点即为结点\*p的后继结点。
- 在中序穿线树中找寻某结点\*p的前驱的运算与找寻后继的运算是对称的,左改右,右改左,后继改前驱即可。因此有人称中序为对称序。

### 8.1.7 图

#### 1. 图的基础知识

##### 内容要点

图是由顶点集合及顶点间的关系集合(亦称为边集合)组成的一种数据结构,记为  $\text{Graph} = (V, E)$ 。其中,  $V$  是顶点的有穷非空集合,  $E$  是边的有穷集。

图分为两类:有向图与无向图。在有向图中,顶点对  $\langle x, y \rangle$  是有序的,它称为从顶点  $x$  到顶点  $y$  的一条有向边。此时,顶点对  $\langle x, y \rangle$  用一对尖括号括起来,  $x$  是有向边的始点,  $y$  是有向边的终点。在无向图中,顶点对  $(x, y)$  是无序的,它称为与顶点  $x$  和顶点  $y$  相关联的一条边,这条边没有特定的方向。一般为了有别于有向图,顶点对用一对圆括号括起来。

如果图中边数达到最大,就成为完全图。在有  $n$  个顶点的无向完全图中,有  $n(n-1)/2$  条边。在有  $n$  个顶点的有向完全图中,有  $n(n-1)$  条边。

与图有关的概念包括边的权、带权图(网络)、邻接顶点、子图、顶点的度、有向图的顶点的入度与出度、路径、路径长度、简单路径、回路(环)、连通图、连通分量、强连通图、强连通分量、生成树、生成森林。

图的存储主要使用两种:邻接矩阵和邻接表。在图的邻接矩阵表示中,有一个表示各个顶点之间关系的矩阵。若设图  $A$  是一个有  $n$  个顶点的图,则图的邻接矩阵是一个二

维数组 Graph, 它的定义为:

$$\text{Graph}[i][j] = \begin{cases} 1, & \text{if } \langle i, j \rangle \in E \text{ or } (i, j) \in E \\ 0, & \text{otherwise} \end{cases}$$

对于网络 (或带权图), 邻接矩阵定义如下:

$$\text{GraphNet}[i][j] = \begin{cases} W(i, j), & \text{if } i \neq j \text{ and } \langle i, j \rangle \in E \text{ or } (i, j) \in E \\ \infty, & \text{otherwise, but } i \neq j \\ 0, & \text{if } i = j \end{cases}$$

邻接表是邻接矩阵的改进。它把邻接矩阵的  $n$  行改为  $n$  个单链表, 把同一个顶点发出的边链接在同一个称之为边链表的单链表中, 单链表的每一个结点代表一条边, 叫做边结点, 结点中保存有与该边相关联的另一顶点的顶点下标  $\text{dest}$  和指向同一链表中下一个边结点的指针  $\text{link}$ 。如果是带权图时, 结点中还要保存该边上的权值  $\text{cost}$ 。顶点  $i$  的出边表的表头指针  $\text{adj}$  在顶点表的下标为  $i$  的顶点记录中, 该记录还保存了该顶点的其他信息。

图的遍历是指若从已给的连通图中某一顶点出发, 沿着一些边访遍图中所有的顶点, 且使每个顶点仅被访问一次, 就叫做图的遍历。

存在两种遍历连通图的方法: 深度优先查找 DFS 和广度优先查找 BFS。这些方法既适用于无向图, 又适用于有向图。

深度优先查找的思路是: 在访问图中某一起始顶点  $v$  后, 由  $v$  出发, 访问它的任一邻接顶点  $w_1$ ; 再从  $w_1$  出发, 访问与  $w_1$  邻接但还没有访问过的顶点  $w_2$ ; 然后再从  $w_2$  出发, 进行类似的访问, ……如此进行下去, 直至到达所有的邻接顶点都被访问过的顶点  $u$  为止。接着, 退回一步, 退到前一次刚访问过的顶点, 看是否还有其他没有被访问的邻接顶点。如果有, 则访问此顶点, 之后再从此顶点出发, 进行与前述类似的访问; 如果没有, 就再退回一步进行查找。重复上述过程, 直到连通图中所有顶点都被访问过为止。

广度优先查找的思路是: 在访问了起始顶点  $v$  之后, 由  $v$  出发, 依次访问  $v$  的各个未曾被访问过的邻接顶点  $w_1, w_2, \dots, w_i$ , 然后再顺序访问  $w_1, w_2, \dots, w_i$  的所有还未被访问过的邻接顶点。再从这些访问过的顶点出发, 再访问它们的所有还未被访问过的邻接顶点, ……如此做下去, 直到图中所有顶点都被访问到为止。

#### 学习难点

- 在有向图中的边是有向的,  $\langle x, y \rangle$  与  $\langle y, x \rangle$  是不同的两条边。在无向图中的边是无向的,  $(x, y)$  与  $(y, x)$  是同一条边。
- 一般地, 若图  $G$  中有  $n$  个顶点,  $e$  条边, 则

$$e = \frac{1}{2} \left\{ \sum_{i=1}^n \text{TD}(v_i) \right\}$$

- 无向图的邻接矩阵是对称的, 将第  $i$  行的元素值或第  $i$  列的元素值累加起来就



得到顶点  $i$  的度。但有向图的邻接矩阵可能是不对称的。如果第  $i$  行的  $\text{Graph}[i][j] = 1$ , 则表示有一条从顶点  $i$  到顶点  $j$  的有向边, 将第  $i$  行的所有元素值累加起来就得到顶点  $i$  的出度; 如果第  $j$  列的  $\text{Graph}[k][j] = 1$ , 则表示有一条从顶点  $k$  到顶点  $j$  的有向边, 将第  $j$  列的所有元素值累加起来就得到顶点  $j$  的入度。即

$$\text{OD}(i) = \sum_{j=0}^{n-1} \text{Graph}[i][j], \quad \text{ID}(j) = \sum_{k=0}^{n-1} \text{Graph}[k][j].$$

- 对于一个网络的邻接矩阵, 将第  $i$  行的所有权值  $0 < W[i][j] < \infty$  的顶点个数统计出来就得到顶点  $i$  的出度, 将第  $j$  列所有权值  $0 < W[i][j] < \infty$  的顶点个数统计出来就得到顶点  $j$  的入度。
- 使用邻接矩阵表示一个图时, 当图中的边数少于顶点个数时, 邻接矩阵中出现大量的零元素, 如果存储这些零元素, 将耗费大量的存储。因此, 邻接矩阵只适用于图中边数  $e$  与  $n^2$  大致相等的情形。称这样的图为稠密图。
- 在邻接表的边链表中, 各个边结点的链入顺序任意, 视边结点输入次序而定。若设图中有  $n$  个顶点,  $e$  条边, 则用邻接表表示无向图时, 需要  $n$  个顶点结点,  $2e$  个边结点; 用邻接表表示有向图时, 若不考虑逆邻接表, 只需  $n$  个顶点结点,  $e$  个边结点。当  $e$  远远小于  $n^2$  时 (这种图成为稀疏图), 可以节省大量的存储空间。此外, 把同一个顶点的所有边链接在一个单链表中, 可以大大方便图的操作。
- 在无向图的邻接表中可以看到, 每一条边  $(V_i, V_j)$  在邻接表中有两个结点: 一个在  $V_i$  的边链表中, 表示  $(V_i, V_j)$ , 一个在  $V_j$  的边链表中, 表示  $(V_j, V_i)$ 。在较复杂的问题中, 有时需要给被处理的边加标记 (如访问标志或删除标志等), 以免重复处理。必须同时给表示某一条边的两处边结点加标记, 而这两处结点又不在同一个边链表中, 所以很不方便。
- 在执行图的遍历时, 由于图中可能存在回路, 且图的任一顶点都可能与其他顶点相通, 所以在访问完某个顶点之后可能会沿着某些边又回到了曾经访问过的顶点。因此, 为了避免重复访问, 可设置一个标志顶点是否被访问过的辅助数组  $\text{visited}[]$ , 它的初始状态为 0, 在图的遍历过程中, 一旦某一个顶点  $V_i$  被访问, 就立即让  $\text{visited}[V_i]$  为 1, 防止它被多次访问。
- 深度优先查找是利用回溯法对图遍历的过程, 一般利用递归方法实现, 每当向前递归查找某一邻接结点之前, 必须判断该结点是否访问过, 若未访问过, 递归向前访问, 否则跳过该结点, 尝试访问其他邻接结点。
- 广度优先查找是一种分层的查找过程, 每向前走一步可能访问一批顶点, 不像深度优先查找那样有往回退的情况。因此, 广度优先查找不是一个递归的过程, 其算法也不是递归的。为了实现逐层访问, 算法中使用了一个队列, 以记忆正



在访问的这一层和上一层的顶点，以便于向下一层访问。

- 当无向图为非连通图时，从图中某一顶点出发，利用深度优先查找算法或广度优先查找算法不可能遍历到图中的所有顶点，只能访问到该顶点所在的最大连通子图（即连通分量）的所有顶点。若从无向图的每一个连通分量中的一个顶点出发进行遍历，就可以求得无向图的所有连通分量。
- 一个给定的图的邻接矩阵表示是惟一的，但对于邻接表来说，如果边的输入先后次序不同，生成的邻接表表示也不同。因此，对于同样一个图，基于邻接矩阵表示的遍历所得到的 DFS 序列或 BFS 序列是惟一的，基于邻接表表示的遍历所得到的 DFS 序列或 BFS 序列可以是不惟一的。

## 2. 最小代价生成树

### 内容要点

一个连通图的生成树是原图的极小连通子图，它包含原图中的所有顶点，而且有尽可能少的边。这意味着对于生成树来说，若砍去它的一条边，就会使生成树变成非连通图；若给它增加一条边，就会形成图中的一个回路。

对于一个连通网（即连通带权图），生成树不同，每棵树的权（即树中所有边上的权值总和）也可能不同。按照生成树的定义， $n$  个顶点的连通网络的生成树有  $n$  个顶点、 $n-1$  条边。因此，构造最小代价生成树的准则有三条：

- (1) 必须只使用该网络中的边来构造最小代价生成树；
- (2) 必须使用且仅使用  $n-1$  条边来联结网络中的  $n$  个顶点；
- (3) 不能使用产生回路的边。

普里姆算法的基本思想是：从连通网络  $N = \{V, E\}$  中的某一顶点  $u_0$  出发，选择与它关联的具有最小权值的边  $(u_0, v)$ ，将其顶点加入到生成树的顶点集合  $U$  中。以后每一步从一个顶点在  $U$  中，而另一个顶点不在  $U$  中的各条边当中选择权值最小的边  $(u, v)$ ，把它的顶点加入到集合  $U$  中，这意味着边  $(u, v)$  也加入到生成树的边集合中。如此继续下去，直到网络中的所有顶点都加入到生成树顶点集合  $U$  中为止。

### 学习难点

- 使用不同的遍历图的方法，可以得到不同的生成树；从不同的顶点出发，也可能得到不同的生成树。
- 构造最小代价生成树的方法有多种，典型的有两种：Kruskal 算法和 Prim 算法。它们都采用了一种逐步求解（Grady）的策略：

设有一个连通网络  $N = \{V, E\}$ ，顶点集合  $V$  中有  $n$  个顶点。最初先构造一个包括全部  $n$  个顶点和 0 条边的森林  $F = \{T_0, T_1, \dots, T_{n-1}\}$ ，以后每一步向  $F$  中加入一条边，它应当是一端在  $F$  的某一棵树  $T_i$  上，而另一端不在  $T_i$  上的所有边中具有最小权值的边。由于边的加入，使  $F$  中的某两棵树合并为一棵，树的棵数减一。经过  $n-1$  步，最终得到一棵有  $n-1$  条边的各边权值总和达到最小的最

小生成树。

- 如果连通网络中各边上的权值互不相等,构造出来的最小代价生成树是惟一的;如果存在权值相等的边,由于选择的次序不同,构造出来的最小代价生成树是不惟一的,不过它们总的权值之和应相同。

### 3. 最短路径

#### 内容要点

在一个不带权的图中,若从一顶点到另一顶点存在着一条路径(仅限于无回路的简单路径),则称该路径的长度为该路径上经过的边的数目,它等于该路径上的顶点数减1。由于从一顶点到另一顶点可能存在着多条路径,每条路径上所经过的边数可能不同,即路径长度不同,则称路径长度最短(即经过的边数最少)的那条路径为最短路径,其路径长度叫做最短路径长度或最短距离。

上面所述的图的最短路径问题只是对无权图而言,若图是带权图,则把从一个顶点  $i$  到图中其余任一顶点  $j$  的一条路径上所经过边上的权值之和定义为该路径的带权路径长度,从  $v_i$  到  $v_j$  可能不止一条路径,我们把带权路径长度最短(即其值最小)的那条路径也称作最短路径,其权值也称作最短路径长度或最短距离。

求图的最短路径问题包括两个方面:一是求图中一顶点到其余各顶点的最短路径,二是求图中每对顶点之间的最短路径。

(1) 为求图中一顶点到其余各顶点的最短路径,常用 Dijkstra 算法。该算法采用邻接矩阵作为图的存储表示。在构造过程中,还设置了两个辅助数组:

$lowcost[]$ : 存放生成树顶点集合内顶点到生成树外各顶点的边上的当前最小权值;

$nearestx[]$ : 记录生成树顶点集合外各顶点距离集合内哪个顶点最近(即权值最小)。

若选择从顶点 0 出发,即  $u_0 = 0$ ,因为在生成树顶点集合内最初只有一个顶点 0,故在  $nearestx$  数组中,只有表示顶点 0 的数组元素  $nearestx[0] = -1$ ,其他都是 0,表示集合外各顶点  $i$  ( $0 < i < n$ ) 距离集合内最近的顶点是 0。数组  $lowcost[i]$  的内容都是从邻接矩阵的第 0 行复制来的。然后反复做以下工作:

① 在  $lowcost[]$  中选择  $nearestx[i] \neq -1$  且  $lowcost[i]$  最小的边  $i$ ,用  $v$  标记它。则选中的权值最小的边为  $(nearestx[v], v)$ ,相应的权值为  $lowcost[v]$ 。

② 将  $nearestx[v]$  改为  $-1$ ,表示它已加入生成树顶点集合。将边  $(nearestx[v], v, lowcost[v])$  加入生成树的边集合。

③ 取  $lowcost[i] = \min \{ lowcost[i], Edge[v][i] \}$ ,即用生成树顶点集合外各顶点  $i$  到刚加入该集合的顶点  $v$  的距离  $Edge[v][i]$  与原来  $i$  到生成树顶点集合中顶点的最短距离  $lowcost[i]$  做比较,取距离近的,作为这些集合外顶点到生成树顶点集合内顶点的最短距离。

④ 如果生成树顶点集合外顶点  $i$  到刚加入该集合的顶点  $v$  的距离比原来它到生成树顶点集合中顶点的最短距离还要近,则修改  $nearestx[i] = v$ 。表示生成树外顶点  $i$  到生成

树内顶点  $v$  当前距离最近。

(2) 所有顶点之间的最短路径问题的提法是：已知一个各边权值均大于 0 的带权有向图，对每一对顶点  $v_i \neq v_j$ ，要求求出  $v_i$  与  $v_j$  之间的最短路径和最短路径长度。

Floyd 算法使用邻接矩阵  $\text{Graph}[n][n]$  存储带权有向图。算法的基本思想是：设置一个  $n \times n$  的方阵  $A^{(k)}$ ，其中除对角线的矩阵元素都等于 0 外，其他元素  $a^{(k)}[i][j]$  ( $i \neq j$ ) 表示从顶点  $v_i$  到顶点  $v_j$  的有向路径长度， $k$  表示运算步骤。初始时，以任意两个顶点之间的直接有向边的权值作为路径长度：对于任意两个顶点  $v_i$  和  $v_j$ ，若它们之间存在有向边，则以此边上的权值作为它们之间的最短路径长度；若它们之间不存在有向边，则以 MAXNUM（机器可表示的在问题中不会遇到的最大数）作为它们之间的最短路径长度。因此， $A^{(-1)} = \text{Graph}$ 。以后逐步尝试在原路径中加入其他顶点作为中间顶点。如果增加中间顶点后，得到的路径比原来的路径长度减少了，则以此新路径代替原路径，修改矩阵元素，代入新的更短的路径长度。

#### 学习难点

- 带权有向图  $D$  的某几条边或所有边的长度可能为负值。利用 Dijkstra 算法，不一定能得到正确的结果。因此，限定算法中处理的边的权值不能是负数或 0。
- 解决所有顶点之间的最短路径问题的另一个办法是：轮流以每一个顶点为源点，重复执行迪克斯特拉算法  $n$  次，就可求得每一对顶点之间的最短路径及最短路径长度。

- Floyd 算法可描述如下：

定义一个  $n$  阶方阵序列： $A^{(-1)}, A^{(0)}, \dots, A^{(n-1)}$ ，其中：

$$A^{(-1)}[i][j] = \text{Graph}[i][j];$$

$$A^{(k)}[i][j] = \min \{ A^{(k-1)}[i][j], A^{(k-1)}[i][k] + A^{(k-1)}[k][j] \}, k = 0, 1, \dots, n-1$$

上述的公式是系列递推公式，其中， $A^{(0)}[i][j]$  是从顶点  $v_i$  到  $v_j$ ，中间顶点是  $v_0$  的最短路径的长度， $A^{(k)}[i][j]$  是基于  $A^{(k-1)}[i][j]$  的从顶点  $v_i$  到  $v_j$ ，中间顶点的序号不大于  $k$  的最短路径的长度， $A^{(n-1)}[i][j]$  是从顶点  $v_i$  到  $v_j$  的最短路径长度。

- Floyd 算法允许图中有带负权值的边，但不许有包含带负权值的边组成的回路。
- 上面给出的求解最短路径的算法不仅适用于带权有向图，对带权无向图也可以适用。因为带权无向图可以看做是有往返二重边的有向图，只要在顶点  $v_i$  与  $v_j$  之间存在无向边  $(v_i, v_j)$ ，就可以看成是在这两个顶点之间存在权值相同的两条有向边  $\langle v_i, v_j \rangle$  和  $\langle v_j, v_i \rangle$ 。

#### 4. 拓扑排序

##### 内容要点

设  $S$  是一个集合， $R$  是  $S$  上的一个关系。若  $R$  中的关系满足传递性和非自反性，则  $R$  是  $S$  上的半序关系。注意， $S$  中的某些元素不全满足  $R$ 。如果通过人为地加入一些关系，使得在  $S$  中所有元素都满足  $R$ ，这就是拓扑排序。拓扑排序的结果可得拓扑有序的



序列。

如果用有向图表示一个工程,用顶点表示活动,用有向边 $\langle V_i, V_j \rangle$ 表示活动 $V_i$ 必须先于活动 $V_j$ 进行。这种有向图叫做顶点表示活动的网络,记作AOV网络。

在AOV网络中,如果活动 $V_i$ 必须在活动 $V_j$ 之前进行,则存在有向边 $\langle V_i, V_j \rangle$ ,并称 $V_i$ 是 $V_j$ 的直接前驱, $V_j$ 是 $V_i$ 的直接后继。这种前驱与后继的关系有传递性。此外,任何活动 $V_i$ 不能以它自己作为自己的前驱或后继,这叫做反自反性。从前驱和后继的传递性和反自反性来看,AOV网络中不能出现有向回路,即有向环。

检测有向环的方法是对AOV网络构造它的拓扑有序序列。即将各个顶点(代表各个活动)排列成一个线性有序的序列,使得AOV网络中所有应存在的前驱和后继关系都能得到满足。这种构造AOV网络全部顶点的拓扑有序序列的运算就叫做拓扑排序。如果通过拓扑排序能将AOV网络的所有顶点都排入一个拓扑有序的序列中,则该AOV网络中必定不会出现有向环;相反,如果得不到满足要求的拓扑有序序列,则说明AOV网络中存在有向环。

拓扑排序的方法:

① 输入AOV网络。令 $n$ 为顶点个数。

② 在AOV网络中选一个没有直接前驱的顶点,并输出之;

③ 从图中删去该顶点,同时删去所有它发出的有向边;

重复以上②、③步,直到

a. 全部顶点均已输出,拓扑有序序列形成,拓扑排序完成;

或 b. 图中还有未输出的顶点,但已跳出处理循环。这说明图中还剩下一些顶点,它们都有直接前驱,再也找不到没有前驱的顶点了。这时AOV网络中必定存在有向环。

学习难点

- 在实际编写拓扑排序算法时,建立了一个存放入度为零的顶点的链式栈,供选择和输出无前驱的顶点。只要出现入度为零的顶点,就将它加入栈中。使用这种栈的拓扑排序算法可以描述如下:

① 建立入度为零的顶点栈;

② 当入度为零的顶点栈不空时,重复执行 {

从顶点为0的顶点栈中退出一个顶点,并输出之;

从AOV网络中删去这个顶点和它发出的边,边的终顶点入度减一;

如果边的终顶点入度减至0,则该顶点进入度为零的顶点栈;

}

③ 如果输出顶点个数少于AOV网络的顶点个数,则报告网络中存在有向环。

- 如果从一个结点有多个直接后继,则拓扑排序的结果可以不惟一。
- 由于AOV网络中各顶点的地位是平等的,每个顶点的编号是人为的,因此可



以按照拓扑排序的结果重新安排顶点的序号,生成 AOV 网络的新的邻接矩阵存储表示。在这种邻接矩阵中,对角线以下可以全零。

## 5. 关键路径

### 内容要点

与 AOV 网络密切相关的另一种网络就是 AOE 网络。如果在无有向环的带权有向图中用有向边表示一个工程中的各项活动,用有向边上的权值表示活动的持续时间,用顶点表示事件,则这样的有向图叫做用边表示活动的网络,简称 AOE 网络。

在 AOE 网络中,有些活动可以并行地进行。从始点到各个顶点,以至从始点到终点的有向路径可能不止一条。这些路径的长度也可能不同。完成不同路径的活动所需的时间虽然不同,但只有各条路径上所有活动都完成了,整个工程才算完成。因此,完成整个工程所需的时间取决于从始点到终点的最长路径长度,即在这条路径上所有活动的持续时间之和。这条路径长度最长的路径就叫做关键路径。

要找出关键路径,必须找出关键活动,即不按期完成就会影响整个工程完成的活动。关键路径上的所有活动都是关键活动。因此,只要找到了关键活动,就可以找到关键路径。

事件  $V_i$  的最早可能开始时间  $Est(i)$  是从始点  $V_0$  到顶点  $V_i$  的最长路径长度。求  $Est[i]$  的公式从  $Est[0] = 0$  开始,向前递推

$$Est[i] = \max_j \{ Est[j] + \text{dur}(\langle V_j, V_i \rangle) \}, \quad \langle V_j, V_i \rangle \in S2, \quad i=1,2,\dots,n-1$$

其中,  $S2$  是所有指向顶点  $V_i$  的有向边  $\langle V_j, V_i \rangle$  的集合。

事件  $V_i$  的最迟允许开始时间  $Elt[i]$  是在保证终点  $V_{n-1}$  在  $Est[n-1]$  时刻完成的前提下,事件  $V_i$  的允许的最迟开始时间。它等于  $Est[n-1]$  减去从  $V_i$  到  $V_{n-1}$  的最长路径长度。求  $Elt[i]$  的公式从  $Elt[n-1] = Est[n-1]$  开始,反向递推

$$Elt[i] = \min_j \{ Elt[j] - \text{dur}(\langle V_i, V_j \rangle) \}, \quad \langle V_i, V_j \rangle \in S1, \quad i=n-2, n-3, \dots, 0$$

其中,  $S1$  是所有从顶点  $V_i$  发出的有向边  $\langle V_i, V_j \rangle$  的集合。

设活动  $a_k$  在有向边  $\langle V_i, V_j \rangle$  上,则  $ast[k]$  是从始点  $V_0$  到顶点  $V_i$  的最长路径长度。因此,  $ast[k] = Est[i]$ 。

设活动  $a_k$  在有向边  $\langle V_i, V_j \rangle$  上,则  $alt[k]$  是在不会引起时间延误的前提下,该活动允许的最迟开始时间。 $alt[k] = Elt[j] - \text{dur}(\langle i, j \rangle)$ 。其中,  $\text{dur}(\langle i, j \rangle)$  是完成  $a_k$  所需的时间。

$alt[k] - ast[k]$  表示活动  $a_k$  的最早可能开始时间和最迟允许开始时间的的时间余量,也叫做松弛时间。 $alt[k] == ast[k]$  表示活动  $a_k$  是没有时间余量的关键活动。

### 学习难点

- 计算  $Est[i]$  和  $Elt[i]$  的递推公式必须分别在拓扑有序及逆拓扑有序的前提下进行。就是说,计算  $Est[i]$  时,  $V_i$  的所有前驱顶点  $V_j$  的  $Est[j]$  都已求出。反之,在计算  $Elt[i]$  时,也必须在  $V_i$  的所有后继顶点  $V_j$  的  $Elt[j]$  都已求出的条件下才

能进行计算。

- 拓扑排序算法只能检测出网络中的有向回路。网络中可能还存在其他问题。比如说,存在从开始顶点无法到达的顶点。当在这样的网络中进行关键路径计算时,将有多个顶点的  $Est[i]$  等于 0。因为整个网络中各活动的持续时间都应大于 0,所以只有开始顶点的  $Est[0]$  可以等于 0。利用关键路径法也可以检测工程中是否存在有这样的问题。
- 如果在关键路径上的任一关键活动出现时间延误,则会影响到整个工程产生时间延误。然而,如果同时存在几条关键路径,任一关键活动加速,不一定能加速整个工程的进度。只有“桥”(即某一处于所有关键路径上的关键活动)的情况例外。

### 8.1.8 排序

#### 内容要点

排序是根据关键字递增或递减的顺序,把数据记录依次排列起来,使一组任意排列的记录变成一组按其关键字线性有序的记录。

如果在记录序列中有两个记录  $R_{[i]}$  和  $R_{[j]}$ , 它们的关键字  $K_{[i]} == K_{[j]}$ , 且在排序之前, 记录  $R_{[i]}$  排在  $R_{[j]}$  前面。如果在排序之后, 记录  $R_{[i]}$  仍在记录  $R_{[j]}$  的前面, 则称这个排序方法是稳定的, 否则称这个排序方法是不稳定的。

排序方法根据在排序过程中数据记录是否完全在内存, 分为两大类: 内排序和外排序。内排序是指在排序期间数据记录全部存放在内存的排序; 外排序是指在排序期间全部记录不能同时存放在内存, 必须根据排序过程的要求, 不断在内、外存之间移动的排序。

排序是经常使用的一种运算, 因此, 排序的时间开销是衡量算法好坏的最重要的标志。排序的时间开销可用算法执行中的数据比较次数与数据移动次数来衡量。有时还要考虑受记录关键字初始排列及记录个数的影响。

(1) 插入排序主要包括直接插入排序和希尔排序两种。

直接插入排序的基本思想是: 把数组  $A[n]$  中待排序的  $n$  个元素看成为一个有序表和一个无序表, 开始时有序表中只包含一个元素  $A[0]$ , 无序表中包含有  $n-1$  个元素  $A[1] \sim A[n-1]$ , 排序过程中每次从无序表中取出第一个元素, 把它插入到有序表中的适当位置, 使之成为新的有序表, 这样经过  $n-1$  次插入后, 无序表就变为空表, 有序表中就包含了全部  $n$  个元素, 至此排序完毕。在有序表中寻找插入位置时采用从后向前的顺序查找的方法。

用线性链表实现直接插入排序的思想也是一样的。设有两个链表, 一个是初始只有一个元素的有序链表, 一个是待排序元素构成的链表。在有序链表中寻找插入位置和链表插入操作与线性链表相同, 当待排序链表的全部元素都取空并插入到有序链表后排序

完成。

希尔排序又称缩小增量排序,它是对直接插入排序的一种改进方法,是由 D.L.Shell 于 1959 年提出的。希尔排序的过程是:首先以  $d_1 (0 < d_1 < n-1)$  为步长,把数组 A 中  $n$  个元素分为  $d_1$  个组,使下标距离为  $d_1$  的元素在同一组中,即  $A[0], A[d_1], A[2d_1], \dots$  为第一组,  $A[1], A[1+d_1], A[1+2d_1], \dots$  为第二组,  $\dots, A[d_1-1], A[2d_1-1], A[3d_1-1], \dots$  为最后一组,对每一组进行直接插入排序;然后再以  $d_2 (d_2 < d_1)$  为步长,在上一步排序的基础上,把 A 中的  $n$  个元素重新分为  $d_2$  个组,使下标距离为  $d_2$  的元素在同一组中,在每一组内进行直接插入排序;照此办理,直到  $d_i = 1$ ,以  $d_i$  为步长,把所有  $n$  个元素看做为 1 组,进行直接插入排序,最终完成整个排序。

(2) 选择排序主要包括直接选择排序和堆排序两种。

直接选择排序每次从待排序的区间中选择出具有最小关键字的元素,将该元素与该区间的第一个元素交换位置。第一趟(即开始)待排序区间包含所有元素  $A[0] \sim A[n-1]$ ,经过选择和交换后,  $A[0]$  为具有最小关键字的元素;第二趟待排序区间为  $A[1] \sim A[n-1]$ ,经过选择和交换后,  $A[1]$  为仅次于  $A[0]$  的具有最小关键字的元素;第三趟待排序区间为  $A[2] \sim A[n-1]$ ,经过选择和交换后,  $A[2]$  为仅次于  $A[0]$  和  $A[1]$  的具有最小关键字的元素;照此办理,经过  $n-1$  趟选择和交换后,  $A[0] \sim A[n-1]$  就成为了有序表,整个排序过程结束。

用线性链表实现的直接选择排序算法的思想是同时设置两个链表,一个是原始链表,存放所有待排序记录,另一个是结果链表,存放排好序的记录。每趟在原始链表中摘下关键字最大的结点(几个关键字相等时为最前面的结点),把它插入到结果链表的最前端。由于在原始链表中摘下的关键字越来越小,在结果链表前端插入的关键字也越来越小,最后形成的结果链表中的结点将按关键字非递减的顺序有序链接。

堆排序是利用堆的特性进行排序的过程。堆是这样一棵顺序存储的完全二叉树:若从 0 开始对树中的结点按层、同一层再按从左到右的次序进行编号,则编号为  $0 \sim \lfloor n/2 \rfloor - 1$  的结点为分支结点,编号大于  $\lfloor n/2 \rfloor - 1$  的结点为叶结点,对于每个编号为  $i$  的分支结点,它的左子女结点的编号为  $2i+1$ ,它的右子女结点的编号为  $2i+2$ ;除编号为 0 的树根结点外,对于每个编号为  $i$  的结点,它的双亲结点的编号为  $\lfloor (i-1)/2 \rfloor$ 。假定结点  $i$  中存放记录的关键字为  $S_i$ ,若此二叉树各结点中的关键字满足下列特性则称它为堆。

$$S_i \geq S_{2i+1} \text{ 和 } S_i \geq S_{2i+2} \quad (0 \leq i \leq \lfloor n/2 \rfloor - 1)$$

堆排序包括构成初始堆和利用堆排序两个阶段。

构成初始堆就是把待排序的元素集合  $\{R_0, R_1, \dots, R_{n-1}\}$ ,按照堆的定义调整为堆  $\{R'_0, R'_1, \dots, R'_{n-1}\}$ ,其中对应的关键字  $S'_i \geq S_{2i+1}$  和  $S'_i \geq S_{2i+2}$ ,  $0 \leq i \leq \lfloor n/2 \rfloor - 1$ 。为此需从对应的完全二叉树中编号最大的分支结点(即编号为  $\lfloor n/2 \rfloor - 1$  的结点)起,至根结点(即编号为 0 的结点)止,依次对每个分支结点进行“渗透”运算,形成以该分支结点为根的堆,当最后对二叉树的根结点进行渗透运算后,整个二叉树就构成了一个堆。

从分支结点  $R_i (0 \leq i \leq \lfloor n/2 \rfloor - 1)$  进行渗透运算时,首先把  $R_i$  的关键字  $S_i$  与两个子女结



点中关键字较大者  $S_j$  ( $j = 2i+1$  或  $2i+2$ ) 进行比较, 若  $S_i \geq S_j$ , 则以  $S_i$  为根的子树成为堆, 渗透运算结束, 否则  $R_i$  与  $R_j$  互换位置, 互换后可能破坏以  $R_j$  (此时的  $R_j$  的值为原来的  $R_i$  的值) 为根的堆, 接着再把  $R_j$  与它的两个子女结点中关键字较大者进行比较, 依此类推, 直到双亲结点的关键字大于等于子女结点中较大的关键字或者子女结点为空时止。这样, 以  $R_i$  为根的子树就被调整为一个堆。

根据堆的定义和上面建堆的过程可以知道, 编号为 0 的结点  $A[0]$  (即堆顶) 是堆中  $n$  个结点中关键字最大的结点。所以堆排序的过程比较简单, 首先把  $A[0]$  与  $A[n-1]$  对换, 使  $A[n-1]$  为关键字最大的结点, 接着对  $A[0]$  (即对调前的  $A[n-1]$ ) 在前  $n-1$  个结点中进行渗透运算, 又得到  $A[0]$  为当前集合  $A[0] \sim A[n-2]$  内具有最大关键字的结点, 再接着把  $A[0]$  同当前集合内的最后一个结点  $A[n-2]$  对换, 使  $A[n-2]$  为次最大关键字结点, 这样经过  $n-1$  次对换和渗透运算后, 所有结点都按其关键字排好序, 排序结束。

(3) 交换排序主要包括冒泡排序和快速排序两种。

冒泡排序的基本思想是通过相邻元素之间的比较和交换使关键字较小的元素逐渐从底部移向顶部, 即从下标较大的位置移向下标较小的位置, 就像水底下的气泡一样逐渐向上冒。冒泡排序过程为: 首先将  $A[n-1]$  元素的关键字与  $A[n-2]$  元素的关键字进行比较, 如果发生逆序, 即  $A[n-2]$  元素的关键字  $>$   $A[n-1]$  元素的关键字, 则交换两元素的位置, 使轻者 (即关键字较小的元素) 上浮, 重者 (即关键字较大的元素) 下沉, 接着比较  $A[n-2]$  同  $A[n-3]$  元素的关键字, 同样使轻者上浮, 重者下沉, 依此类推, 直到比较  $A[1]$  同  $A[0]$  元素的关键字, 并使轻者上浮重者下沉后, 第一趟排序结束, 此时  $A[0]$  为具有最小关键字的元素; 然后在  $A[n-1] \sim A[1]$  排序区间内进行第二趟排序, 使次最小关键字的元素被上浮到第 1 单元中; 重复进行  $n-1$  趟后, 整个冒泡排序结束。

快速排序, 就平均性能来讲, 是目前所有排序方法中速度最快的一种。快速排序的过程可叙述为: 首先从待排序区间 (开始时为  $A[0] \sim A[n-1]$ ) 中选取一个元素 (为方便起见, 一般选取该区间的第一个元素) 作为比较的基准, 进行一趟划分, 即通过从区间两端向中间顺序进行比较和交换, 使区间前半部分只保留比基准元素的关键字小的元素, 后半部分只保留比基准元素的关键字大的元素, 当所有元素的关键字都比较过一遍后, 把基准元素放置到前后两部分的交界处, 这样, 前半部分所有元素的关键字均小于等于基准元素的关键字, 后半部分所有元素的关键字均大于等于基准元素的关键字, 基准元素的当前位置就是排序后的最终位置, 然后再对基准元素的前后两个子区间分别进行快速排序。这是一个递归的过程, 递归结束的条件是: 当一个区间为空或只包含一个元素时, 就结束该区间上的快速排序过程。

(4) 合并排序

合并又称归并, 是将两个或多个有序表合并成一个有序表的过程。若将两个有序表合并成一个有序表则称为二路合并。例如有两个有序表 (7,12,15,20) 和 (4,8,10,17), 合并后得到的有序表为 (4,7,8,10,12,15,17,20)。二路合并算法的思路是: 假定待合并的两个



有序表分别存于数组 A 中从下标 s 到下标 m 的单元和从下标 m+1 到下标 t 的单元 ( $s \leq m, m+1 \leq t$ ), 结果有序表存于数组 R 中从下标 s 到下标 t 的单元, 并令 i, j, k 分别指向这些有序表的第一个单元。合并过程为: 比较  $A[i]$  和  $A[j]$  的关键字大小, 若  $A[i]$  的关键字  $\leq A[j]$  的关键字, 则将第一个有序表中的元素  $A[i]$  复制到  $R[k]$  中, 并令 i 和 k 分别加 1, 使之分别指向后一存放位置, 否则将第二个有序表中的元素  $A[j]$  复制到  $R[k]$  中, 并令 j 和 k 分别加 1; 如此循环下去, 直到其中一个有序表比较和复制完, 然后再将另一个有序表中剩余的元素复制到 R 中从下标 k 到下标 t 的单元。

合并排序又称归并排序, 是利用合并操作把一个无序表排列成一个有序表的过程。若利用二路合并操作则称为二路合并排序。二路合并排序的过程是: 首先把待排序区间(即无序表)中的每一个元素都看做为一个有序表, 则 n 个元素构成 n 个有序表; 接着两两合并, 这样就得到了  $\lceil n/2 \rceil$  个长度为 2 (最后一个表的长度可能小于 2) 的有序表, 称此为一趟合并; 然后再两两进行有序表的合并, 得到  $\lceil n/2 \rceil / 2$  个长度为 4 (最后一个表的长度可能小于 4) 的有序表; 如此进行下去, 直到合并第  $\lceil \log_2 n \rceil$  趟后得到一个长度为 n 的有序表为止。

#### (5) 外排序

外排序就是对外存文件中的记录进行排序的过程, 排序结果仍然被放到原有文件中。

基于磁盘进行的排序多使用归并排序方法。其排序过程主要分为两个阶段: 第一个阶段建立为外排序所用的内存缓冲区。根据它们的大小将输入文件划分为若干段, 用某种有效的内排序方法, 对各段进行排序。这些经过排序的段叫做初始归并段。当它们生成后就被写到外存中去。第二阶段仿照内排序中所介绍过的归并树模式, 把第一阶段生成的初始归并段加以归并, 一趟趟地扩大归并段和减少归并段个数, 直到最后归并成一个大归并段(有序文件)为止。

做 k 路平衡归并时, 如果有 m 个初始归并段, 则相应的归并树有  $\lceil \log_k m \rceil + 1$  层, 需要归并  $\lceil \log_k m \rceil$  趟。做内部归并时, 在 k 个对象中选择最小者, 需要顺序比较 k-1 次。每趟归并 n 个记录需要做  $(n-1) \cdot (k-1)$  次比较, S 趟归并总共需要的比较次数为:

$$S \cdot (n-1) \cdot (k-1) = \lceil \log_k m \rceil \cdot (n-1) \cdot (k-1)$$

使用“败者树”从 k 个归并段中选最小者, 只需要约  $\lceil \log_2 k \rceil$  次关键字比较, 对于较大的 k ( $k \geq 6$ ), 可大大减少查找关键字最小的记录时的比较次数。

败者树实际上是一棵完全二叉树。其中每个叶结点存放各归并段在归并过程中当前参加比较的记录, 每个非叶结点记忆它两个子女结点中记录关键字小的结点(即败者)。因此, 根结点中记忆树中当前记录关键字最小的结点。

#### 学习难点

- 虽然稳定的排序方法和不稳定的排序方法排序结果有差别, 但不能说不稳定的排序方法就不好, 各有各的适应场合。

- 排序的过程可以是对数据记录本身进行物理地重排, 经过比较和判断, 将记录移到合适的位置。这时, 数据记录一般都存放在一个顺序的表中, 以便让比较指针前后移动。此外, 还可以给每个记录增加一个链接指针, 在排序的过程中不移动记录或传送数据, 仅通过修改链接指针来改变记录之间的逻辑顺序, 从而达到排序的目的。这时, 通过链表组织待排序的记录。前者称为静态排序, 使用的静态链表结构自始至终不变。后者称为动态排序, 表的结构在排序过程中不断改变。
- 直接插入排序的记录比较和移动次数与记录的初始排列有关, 在最好情况下, 即在排序前记录已按关键字大小从小到大排序, 总的记录比较次数为  $n-1$ , 记录移动次数为 0; 在最坏情况下, 总的记录比较次数和移动次数为  $n^2/2$ ; 在平均情况下的记录比较次数和移动次数约为  $n^2/4$ 。直接插入排序是一种稳定的排序方法。
- 在希尔排序中, 开始步长(增量)较大, 分组较多, 每个组内的记录数较少, 因而记录的比较和移动次数都较少, 且移动距离较远; 越到后来步长越小(最后一步为 1), 分组越少, 每个组内的记录数也越多, 但同时记录次序也越来越接近有序, 因而记录的比较和移动次数也都较少。从理论上和实验上都已证明, 在希尔排序中, 记录的总的比较次数和总的移动次数比直接插入排序时要少得多, 特别是当  $n$  越大时效果越明显。希尔排序是一种不稳定的排序方法。
- 直接选择排序的记录比较次数不受记录的初始排列的影响, 共进行  $n-1$  趟选择和交换, 每趟选择需要比较  $n-i$  次 ( $1 \leq i \leq n-1$ ), 共需比较  $n(n-1)/2$  次。但记录的移动要受记录的初始排列影响, 最好情况下, 不做交换, 移动 0 次记录; 最坏情况下, 需做  $n-1$  次交换, 移动  $3(n-1)$  次记录。直接选择排序是一种不稳定的排序方法。
- 在堆排序中, 当对二叉树  $R_i$  进行渗透运算时, 比它编号大的分支结点都已进行过渗透运算, 即已形成了以各个分支结点为根的堆, 其中包括以  $R_i$  的左、右子女结点  $R_{2i+1}$  和  $R_{2i+2}$  为根的堆。所以, 对  $R_i$  进行渗透运算是在其左、右子树均为堆的基础上实现的。
- 在堆排序中, 形成初始堆需要执行  $\lfloor n/2 \rfloor$  渗透算法, 在基于初始堆的排序过程中需要执行  $(n-1)$  次交换和渗透算法, 每次执行渗透算法向下进行调整的比较和移动次数不超过树的深度  $\lfloor \log_2 n \rfloor$ , 因此, 堆排序的记录比较和移动次数约为  $n \log_2 n$ 。堆排序是一种不稳定的排序方法。
- 冒泡排序的记录比较和移动次数受记录的初始排列影响。如果待排序记录已经按关键字从小到大排好序时, 此算法只执行一趟, 做  $n-1$  次记录比较, 不移动记录。这是最好的情形。最坏的情形是待排序记录初始时按关键字从大到小有序, 把它们全部逆转过来, 算法要执行  $n-1$  趟, 第  $i$  趟 ( $1 \leq i < n$ ) 做  $n-i$  次比较,

执行  $n-i$  次交换。这样总的记录比较和交换次数均为  $n(n-1)/2$ 。冒泡排序是一种稳定的排序方法。

- 在快速排序中,若把每次划分所用的基准元素看做为根结点,把划分得到的左区间和右区间看做为根结点的左子树和右子树,那么整个排序过程就对应一棵具有  $n$  个元素的二叉查找树,所需划分的层数就等于对应二叉查找树的深度,所需划分的所有区间数,它包括开始非递归调用使用的区间和每次递归调用所使用的区间的总和等于对应二叉查找树中分支结点数。最好情况下,即每次划分得到的左、右子区间大小大致相等时,快速排序的速度最快,数据比较和移动次数约  $n \log_2 n$ ; 最坏情况下,若待排序区间上的记录已为正序或逆序,对应的二叉查找树成为一棵单支树,数据比较和移动次数降低到  $n^2$ ,在这种情况下需要递归处理  $n-1$  次(含第 0 次递归调用)。为了避免这种情况的发生,一是若事先知道待排序的记录已基本有序(包括正序和逆序),则采用其他排序方法,而不要采用快速排序方法;二是修改上面的快速排序算法,使得在每次划分之前比较当前区间的第一个元素、最后一个元素和中间一个元素的关键字,取关键字居中的一个元素作为基准元素并调换到第一个元素位置。快速排序是一种不稳定的排序方法。

- 二路合并排序的时间代价等于合并趟数与每一趟时间复杂性的乘积。合并趟数为  $\lceil \log_2 n \rceil$  (当  $\lceil \log_2 n \rceil$  为奇数时,则为  $\lceil \log_2 n \rceil + 1$ )。因为每一趟合并就是将两两有序表合并,而每一对有序表合并时,记录的比较次数均小于等于记录的移动次数(即由一个数组复制到另一个数组中的记录个数),而记录的移动次数等于这一对有序表的长度之和,所以每一趟合并的移动次数均等于数组中记录的个数  $n$ 。二路合并排序的时间代价约为  $n \log_2 n$ 。

二路合并排序时需要利用同待排序数组一样大小的一个辅助数组,需要附加空间较多。它是一种稳定的排序方法。

- 一般来讲,对  $m$  个初始归并段,做  $k$  路归并,归并树可用正则  $k$  叉树(即只有度为  $k$  与度为 0 的结点的  $k$  叉树)来表示。第一趟可将  $m$  个初始归并段归并为  $\lceil m/k \rceil$  个归并段,以后每一趟归并将 1 个归并段归并成  $\lceil 1/k \rceil$  个归并段,直到最后形成一个大的归并段为止。树的深度  $= \lceil \log_k m \rceil =$  归并趟数  $S$ 。只要增大归并路数  $k$ ,或减少初始归并段个数  $m$ ,都能减少归并趟数  $S$ ,以减少读写磁盘次数  $d$ ,达到提高外排序速度的目的。
- 为了减少读写磁盘次数,除了增加归并路数  $k$  外,还可以减少初始归并段个数  $m$ 。在总对象数  $n$  一定时,要想减少  $m$ ,必须增大初始归并段的长度。如果规定每个初始归并段等长,则此长度应根据生成它的内存工作区空间大小而定,因而  $m$  的减少也就受到了限制。为了突破这个限制,可采用败者树来生成初始归并段。在使用同样大的内存工作区的情况下,可以生成平均比原来等长情况



下大一倍的初始归并段，从而减少参加多路平衡归并排序的初始归并段个数，降低归并趟数。

### 8.1.9 查找

#### 内容要点

##### (1) 静态查找表的查找

静态查找表是最简单的基于数组的数据表。在静态查找表中，数据记录存放于数组中，利用数组元素的下标作为数据记录的存放地址。查找算法根据给定值  $x$ ，在数组中进行查找。直到找到  $x$  在数组中的存放位置或可确定在数组中找不到  $x$  为止。如果各数据记录具有简单的数据类型，如 `int` 或 `string`，可以直接用  $x$  与数组每个元素中存储的数据进行比较；如果各数据记录具有比较复杂的数据类型，如用户自定义的数据类型，则需要用  $x$  与各数据记录的关键字 `key` 进行比较。

##### ① 无序顺序表上的顺序查找

顺序查找，又称线性查找，主要用于在线性结构中进行查找。设无序顺序表中有  $n$  个对象，则顺序查找从表的先端开始，顺序用各记录的关键字与给定值  $x$  进行比较，直到找到与其值相等的记录，则查找成功，给出该记录在表中的位置。若整个表都已检测完仍未找到关键字与  $x$  相等的记录，则查找失败。给出失败信息。

##### ② 有序顺序表上的顺序查找

如果查找表中各个记录按关键字从小到大或从大到小排好序，形成有序表的话，顺序查找的平均查找长度也能降低。因为不成功的查找不一定要比较到表尾，而只要比较到表中记录的关键字大于给定的值，就能确定表中不存在要找的记录。

##### ③ 有序顺序表上的二分法查找

若设有  $n$  个记录存放在一个有序顺序表中，并按其关键字从小到大排好了序。采用二分法查找时，先求出位于查找区间正中的记录的下标 `mid`，用其关键字与给定值  $x$  进行比较，比较的结果有三种可能性：

- 若位于 `mid` 的记录的关键字等于  $x$ ，则查找成功，报告成功信息并返回其下标；
- 若位于 `mid` 的记录的关键字大于  $x$ ，说明如果表中存在要找的记录，它一定在 `mid` 左侧，可把查找区间缩小到表的左半部分，再进行二分法查找；
- 若位于 `mid` 的记录的关键字小于  $x$ ，说明如果表中存在要找的记录，它一定在 `mid` 右侧，可把查找区间缩小到表的右半部分，再进行二分法查找。

每比较一次，查找区间缩小一半。因此在最坏情况下查找到要求记录所需的关键字比较次数约为  $\log_2 n$ 。对于较大的  $n$ ，显然比顺序查找快得多。如果查找区间已经缩小到一个记录，经与给定值比较仍未找到想要查找的记录，则查找失败。

##### ④ 有序顺序表上的插值查找

插值查找法求中间点公式为



$$\text{mid} = \text{low} + \frac{x - \text{Data}[\text{low}].\text{key}}{\text{Data}[\text{high}].\text{key} - \text{Data}[\text{low}].\text{key}} (\text{high} - \text{low})$$

其中,  $\text{Data}[\text{high}]$ 与  $\text{Data}[\text{low}]$ 分别为有序顺序表中具有最大关键字和最小关键字的记录,一般在查找区间的两端。插值查找的查找方法类似于二分法查找,它的查找性能在关键字分布比较均匀的情况下优于二分法查找。

## (2) 分块查找

当记录数目较大时,可以把所有  $n$  个记录分为  $b$  个子表(块)存放。所有记录可能是按关键字有序地存放在子表中,也可能在子表中无序地存放。对于前者,可在子表内采用二分法查找;对于后者,在子表内只能顺序查找。

所有这些子表,要求做到分块有序,即后一个子表中所有记录的关键字均大于前一个子表中所有记录的关键字。它们都存放在专门为它们开辟的数据区中。另外,再为它们建立一个索引表。索引表中每一表目叫做索引项,它记录了各子表中最大关键字  $\text{max\_key}$  以及该子表在文件中的起始位置  $\text{obj\_addr}$ 。因此,各个索引项在索引表中的序号与各个子表的块号有一一对应的关系:即第  $i$  个索引项是第  $i$  个子表的索引项,  $i = 0, 1, \dots, n$ 。

在对索引顺序结构进行查找时,一般分为两级查找。先在索引表  $\text{ID}$  查找给定值  $K$ , 确定满足  $\text{ID}[i-1].\text{max\_key} < K \leq \text{ID}[i].\text{max\_key}$  的  $i$  值,即待查记录若存在时可能在的子表的序号。然后再在第  $i$  个子表中按给定值查找要求的记录。

索引表是按  $\text{max\_key}$  有序的,且长度也不大,可以二分法查找,也可以顺序查找。各子表内各个记录如果也是按记录关键字有序的,也可以采用二分法查找或顺序查找;如果不是按记录关键字有序,则只能顺序查找。

## (3) 线性链表上的顺序查找

### ① 无序线性链表上的顺序查找

在线性链表上只能从链表首结点出发,使用一个检测指针循链逐个结点检测。如果检测到某一结点,它的关键字等于给定值  $x$ ,则查找成功,报告结点地址和查找成功的信息。如果循链一直检测到链尾,仍未找到要寻找的记录,则查找失败。

### ② 有序线性链表上的顺序查找

在这种情况下的查找方法仍然是循链逐个结点进行比较,如果某个结点上的关键字等于给定值  $x$ ,则查找成功;如果找到某个结点的关键字大于给定值  $x$ ,则可立刻退出检测。报告失败信息。因此,查找不成功不必找到链尾。

## (4) 散列表上的查找

散列方法在表项的存储位置与它的关键字之间建立一个确定的对应函数关系  $\text{Hash}()$ ,使每个关键字与结构中的一个惟一的存储位置相对应:

$$\text{Address} = \text{Hash}(\text{Rec.key})$$

散列方法需要讨论以下两个问题:

- 对于给定的一个关键字集合,选择一个计算简单且地址分布比较均匀的散列函

数, 避免或尽量减少冲突:

- 拟订解决冲突的方案。

常见的散列函数有以下几种。

- ① 除余法: 质数  $p$  应取小于或等于  $m$  的最大质数,  $H(\text{key}) = \text{key} \% p$ 。
- ② 基数转换法: 关键字从某一基数的整数转换为另一基数的整数。
- ③ 平方取中法: 关键字求平方, 取结果的中间若干位作为散列地址。
- ④ 折叠法: 将关键字分成若干段, 各段折叠相加, 取其结果作为散列地址。
- ⑤ 移位法: 将关键字分成若干段, 各段对齐相加, 取其结果作为散列地址。

其中以除余法的随机化程度最高。

解决冲突的方法分两类: 开地址法和拉链法, 开地址法又分为线性探查法和双散列函数法:

#### ① 线性探查法

使用某一种散列函数计算出初始散列地址  $H_0$ , 一旦发生冲突, 在表中顺次向后寻找“下一个”空位  $H_i$  的公式为:

$$H_i = (H_{i-1} + 1) \% m, i = 1, 2, \dots, m-1$$

即用线性探查序列  $H_0 + 1, H_0 + 2, \dots, m-1, 0, 1, 2, \dots, H_0 - 1$  在表中寻找“下一个”空位的地址。它亦可写成

$$H_i = (H_0 + i) \% m, i = 1, 2, \dots, m-1$$

每当发生冲突后, 就探查下一个位置。当循环  $m-1$  次后就会回到开始探查时的位置, 说明待查关键字不在表内, 而且表已满, 不能再插入新关键字。

#### ② 双散列函数法

使用双散列函数法时, 需要两个散列函数。第一个散列函数  $\text{Hash}()$  按表项的关键字  $\text{key}$  计算表项的初始散列地址  $H_0 = \text{Hash}(\text{key})$ 。一旦发生冲突, 利用第二个散列函数  $\text{ReHash}()$  计算该表项到达“下一个”空位的跳跃量。它的取值与  $\text{key}$  的值有关, 要求它的取值应当是小于地址空间大小  $m$ , 且与  $m$  互质的正整数。

若设表的长度为  $m$ , 则在表中寻找“下一个”空位的公式为:

$$j = H_0 = \text{Hash}(\text{key}), p = \text{ReHash}(\text{key});$$

$$j = (j + p) \% m; p \text{ 是小于 } m \text{ 且与 } m \text{ 互质的整数}$$

这是为了解决线性探查法处理冲突容易产生“堆积”的问题。利用双散列法, 按一定的距离, 跳跃式地寻找“下一个”空位, 减少了“堆积”的机会。

双散列法的探查序列也可写成:

$$H_i = (H_0 + i * \text{ReHash}(\text{key})) \% m, i = 1, 2, \dots, m-1$$

最多经过  $m-1$  次探查, 它会遍历表中所有位置, 回到  $H_0$  位置。

#### ③ 拉链法

拉链法首先对关键字集合用某一个散列函数计算它们的存放位置。若设散列表地址

空间的所有位置是从 0 到  $m-1$ , 则关键字集合中的所有关键字被划分为  $m$  个子集合, 通过散列函数计算出来的具有相同地址的关键字归于同一子集合。称同一子集合中的关键字互为同义词。每一个子集合也称为一个桶。通常每一个桶中的表项通过一个单链表链接起来, 亦称之为同义词子表。所有桶号相同的表项都链接在同一个同义词子表中, 各链表的表头结点组成一个向量。因此, 向量的元素个数与可能的桶数一致。桶号为  $i$  的同义词子表的表头结点是向量中的第  $i$  个元素。

采用拉链法处理冲突, 每个桶中的同义词子表都很短, 设有  $n$  个关键字通过某一个散列函数, 存放到散列表中的  $m$  个桶中。每一个桶中的同义词子表的平均长度为  $n/m$ , 以查找平均长度为  $n/m$  的同义词子表代替了查找长度为  $n$  的顺序表, 查找速度快得多。

散列表的装载因子  $\alpha$  表明了一个表中的装满程度。它越大, 说明表越满, 再插入新元素时发生冲突的可能性就越大。而散列表的查找性能, 即平均查找长度依赖于散列表的装载因子, 不直接依赖于  $n$  或  $m$ 。不论表的长度有多大, 我们总能选择一个合适的装载因子, 以把平均查找长度限制在一定范围内。

#### 学习难点

- 在查找成功时的平均查找长度, 是指为确定记录在查找表中的位置所执行的关键字比较次数的期望值。对于一个含有  $n$  个记录的无序顺序表, 在等概率下的查找成功时的平均查找长度是  $(n+1)/2$ ; 查找不成功是指对整个查找表都查找一遍但未找到其关键字与给定值相等的记录, 其平均查找长度为  $n$ 。
- 一般地, 对于有  $n$  个关键字的有序顺序表, 使用二分法查找所需的与给定值进行的关键字比较最多为  $\lceil \log_2(n+1) \rceil$ 。那么, 二分法查找的平均查找长度是

$$ASL_{succ} = \frac{n+1}{n} \log_2(n+1) - 1 \approx \log_2(n+1) - 1$$

- 分块查找的查找成功时的平均查找长度  $ASL_{IndexSeq} = ASL_{Index} + ASL_{SubList}$ , 其中,  $ASL_{Index}$  是在索引表中查找子表位置的平均查找长度,  $ASL_{SubList}$  是在子表内查找对象位置的查找成功的平均查找长度。

设把长度为  $n$  的表分成均等的  $b$  个子表, 每个子表  $s$  个对象, 则  $b = \lceil n/s \rceil$ 。又设表中每个对象的查找概率相等, 若用顺序查找确定对象所在的子表, 则分块查找的查找成功时的平均查找长度为

$$ASL_{IndexSeq} = (b+1)/2 + (s+1)/2 = (b+s)/2 + 1, \quad b = \lceil n/s \rceil$$

分块查找的平均查找长度不仅与表中的对象个数  $n$  有关, 而且与每个子表中的对象个数  $s$  有关。在给定  $n$  的情况下, 当  $s = \sqrt{n}$  时,  $ASL_{IndexSeq}$  取极小值  $\sqrt{n} + 1$ 。这个值比顺序查找强, 但比折半查找差。但如果子表存放在外存时, 还要受到页块大小的制约。

若采用折半查找确定对象所在的子表, 则查找成功时的平均查找长度为

$$ASL_{IndexSeq} = ASL_{Index} + ASL_{SubList} \approx \log_2(b+1) - 1 + (s+1)/2 \approx \log_2(1+n/s) + s/2$$

- 若设检测指针为  $p$ , 其前驱指针为  $q$ , 初始时让  $p$  指向链表的首结点,  $q = \text{NULL}$ ,

每当  $p$  所指结点的关键字不等于给定值  $x$  时, 让  $pr$  记忆  $p$  所指结点的地址,  $p$  循链进到下一结点。在查找成功时,  $p$  指示关键字等于给定值  $x$  的结点,  $pr$  指示  $p$  所指结点的前驱结点。在查找不成功时, 新结点可以插在  $pr$  与  $p$  之间。

- 在构造散列函数时有几点需要加以注意:

其一是散列函数的定义域必须包括需要存储的全部关键字, 而如果散列表允许有  $m$  个地址时, 其值域必须在 0 到  $m-1$  之间。

其二是散列函数计算出来的地址应能均匀分布在整个地址空间中: 若  $key$  是从关键字集合中随机抽取的一个关键字, 散列函数应能以同等概率取 0 到  $m-1$  中的每一个值。

其三是散列函数应是简单的, 能在较短的时间内计算出结果。

- 线性探查方法容易产生“堆积”的问题。即不同探查序列的关键字占据了可利用的空位置, 使得为寻找某一关键字需要经历不同的探查序列的元素, 导致查找时间增加。
- 在线性探查或双散列的情形下不能随便物理删除表中已有的表项。因为若删除表项会影响其他表项的查找, 中断散列表中已有表项的查找链, 导致错误地判断表中找不到那些已有的表项。所以若想删除一个表项时, 只能给它做一个删除标记, 进行逻辑删除。但这样做的副作用是: 在执行多次删除后, 表面上看起来散列表很满, 实际上有许多位置没有利用。因此, 当散列表经常变动时, 最好不用线性探查或双散列法处理冲突, 可改用拉链法来处理冲突。
- 再散列函数  $Rehash()$  的取法很多。例如, 当  $m$  是质数时, 可定义  $ReHash(key) = key \% (m-2) + 1$  或  $ReHash(key) = \lfloor key / m \rfloor \% (m-2) + 1$ , 等。当  $m$  是 2 的方幂时,  $ReHash(key)$  可取从 0 到  $m-1$  中的任意一个奇数。
- 实验表明, 拉链法优于线性探查法或双散列法; 在散列函数中, 用除余法作散列函数优于其他类型的散列函数, 最差的是折叠法。当装载因子  $\alpha$  较高时, 因选择的散列函数不同, 散列表的查找性能差别很大。

## 8.2 例题分析

【例 8-1】在排序算法中每一项都与其他诸项进行比较, 计算出小于该项的个数, 以确定该项的位置叫 A。

散列函数有一个共同性质, 即函数值应当以 B 取其值域的每个值。

设有两个串  $p$  和  $q$ , 其中  $q$  是  $p$  的子串。把  $q$  在  $p$  中首次出现的位置作为子串  $q$  在  $p$  中的位置的算法称为 C。

如果要求一个线性表既能较快地查找, 又能适应动态变化的要求, 则可采用 D 的方法。



算法的计算量的大小称为计算的E。

可选答案:

- |           |        |        |        |
|-----------|--------|--------|--------|
| A: ① 插入排序 | ② 交换排序 | ③ 选择排序 | ④ 枚举排序 |
| B: ① 最大概率 | ② 最小概率 | ③ 平均概率 | ④ 同等概率 |
| C: ① 联接   | ② 匹配   | ③ 求子串  | ④ 求串长  |
| D: ① 分块   | ② 顺序   | ③ 二分法  | ④ 基于属性 |
| E: ① 现实性  | ② 难度   | ③ 复杂性  | ④ 效率   |

正确答案: A: ④ B: ④ C: ② D: ① E: ③

分析:

排序有插入排序、交换排序、选择排序、归并排序、基数排序等多种, 题目所述为枚举排序。

均匀的散列函数、无论其具体构造如何, 都必须以同等概率取其值域的每个值。这样才能使散列地址均匀分布到整个地址空间, 使冲突减少。

子串的定位运算称为串的模式匹配, 亦称匹配。

分块查找又名索引顺序查找, 即先找到元素所在的块, 再从块中找出待查元素。它只要求表内元素逐段有序, 而不要求块内元素有序, 所以查找速度比顺序查找块, 又能适应动态变化。

算法的计算量的大小, 称为计算的时间复杂度。与此类似, 空间复杂度反映算法所需存储量的大小。单讲复杂度, 一般指时间复杂度。

【例 8-2】从下列有关树的叙述中, 选出 5 条正确的叙述。

(1) 二叉树中每个结点有两个子女结点, 而对一般的树则无此限制, 因此二叉树是树的特殊情形。

(2) 当  $k \geq 1$  时, 高度为  $k$  的二叉树至多有  $2^{k-1}$  个结点。

(3) 用二叉树的前序遍历和中序遍历可以导出二叉树的后序遍历。

(4) 穿线树的优点是便于在中序下查找前驱结点和后继结点。

(5) 将一棵树转换成二叉树后, 根结点没有左子树。

(6) 一棵含有  $n$  个结点的完全二叉树, 它的高度是  $\lceil \log_2 n \rceil$ 。

(7) 在二叉树中插入结点, 该二叉树便不再是二叉树。

(8) 采用二叉链表作树的存储结构, 树的前序遍历和其相应的二叉树的前序遍历的结果是一样的。

(9) Huffman 树是带权路径长度最短的树, 路径上权值较大的结点离根较近。

(10) 用一维数组存储二叉树时, 总是以前序遍历顺序存储结点。

正确答案: 正确的有 (3)、(4)、(6)、(8)、(9)

分析:

(1) 这个叙述是错误的, 二叉树不是树的特殊情形。二叉树不但规定每个结点只能

有两棵子树，而且结点的子树要区分为左子树和右子树，即使在结点只有一棵非空子树的情况下也要明确指出该子树是左子树还是右子树，而树没有这样的规定。例如下图中是两棵不同的二叉树，但如果作为树，它们就是相同的了。



(2) 这个叙述是错误的。当根结点所在层次为 0 时，二叉树的第  $k$  层至多有  $2^k$  个结点，而高度为  $k$  的二叉树的最大结点个数为

$$k = \sum_{i=0}^k 2^i = 2^{k+1} - 1$$

(3) 这个叙述是正确的。由二叉树结点的前序遍历序列和中序遍历序列可以推出整个二叉树的结构，进而得到二叉树结点的后序遍历序列，二叉树前序遍历的次序是根、左、右，中序遍历的次序是左、根、右。前序序列的第一个结点即为根结点。在中序序列中，根结点前面是左子树结点的中序序列，根结点后面是右子树结点的中序序列。以这两个序列为依据，又能在前序序列中划分出根的左子树结点的前序序列和右子树结点的前序序列。现在整个二叉树的根结点确定了，而且根的左、右子树的结点前序序列和中序序列也划分出来了。就可以继续运用上述方法求左、右子树的根，直至每个结点在二叉树中的位置。例如，已知某二叉树的结点前序序列为

ABCDE (1)

结点中序序列为

CBDAE (2)

由(1)知根结点为 A，从而在(2)中划分出左子树结点中序序列

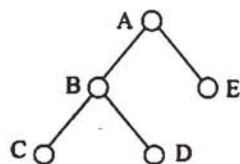
CBD (3)

右子树结点中序序列 E，(右子树只有一个结点因而 E 就是右子树的根)。再回到(1)，可以确定根的左子树结点前序序列

BCD (4)

继续运用上面的方法，由(4)和(3)可确定左子树的根为 B，其左子树包含结点 C，右子树包含结点 D。于是整个二叉树的结构如图所示。

进而可得二叉树结点的后序遍历序列为 CDBEA。类似地，由二叉树结点后序遍历序列和中序遍历序列也可以导出二叉树的前序遍历序列。但由前序遍历序列和后序遍历序列不能导出中序遍历序列，请读者想一想，为什么？



(4) 这个叙述是正确的。一般所说的穿线树指的是中序穿

线树, 它利用结点中空的中左指针位置存储指向结点在中序下的前驱结点的指针 (称为前驱线索), 利用结点中空的中右指针位置存储指向结点在中序下的后继结点的指针, 从而使查找中序下的前驱结点和后续结点变得容易。

(5) 这个叙述是错误的。在树林转换成的二叉树中, 根结点对应树林中第一棵树的根, 左子树对应第一棵树根的子树, 右子树对应树林中其他的树。因此, 将一棵树转换成二叉树后, 应该是根结点的右子树为空。

(6) 这个叙述是正确的。完全二叉树是这样的二叉树: 除层次最大的一层外, 其他各层都达到最大结点数, 而且在层次最大的一层中, 结点都在该层的最左边若干个位置上。由完全二叉树的定义和上面关于②的讲解知道, 设完全二叉树的高度为  $K$ , 则

$$2^K - 1 < n \leq 2^{K+1} - 1$$

或

$$2^K \leq n < 2^{K+1}$$

于是

$$K \leq \log_2 n < K+1$$

由于  $K$  是整数, 所以  $K = \lfloor \log_2 n \rfloor$

(7) 这个叙述是错误的。往二叉树中插入新结点, 完全能保持插入结果仍为二叉树结构, 通常将新结点插到适当结点的空的左子女或右子女的位置上。

(8) 这个叙述是正确的。实际上, 不管采用什么存储结构, 树的前序遍历和其相应的二叉树的前序遍历的结果都是一样的。这是由树和二叉树之间的对应关系, 以及树的前序遍历定义和二叉树的前序遍历定义决定的。

(9) 这个叙述是正确的。二叉树的带权路径长度指的是二叉树中所有带权叶结点的带权路径长度之和, 记作

$$\sum_{k=1}^n w_k l_k$$

其中  $n$  是叶结点个数,  $w_k$  和  $l_k$  分别为第  $k$  个叶结点的权值和路径长度。Huffman 树可用于设计使电文总长度最短的二进制前缀编码, 以及安排外排序的最佳合并次序等。

(10) 这个叙述是错误的。用一维数组存储完全二叉树时, 结点的顺序是从根开始, 逐层从左到右, 依次存储各个结点。若用一维数组存储一般二叉树, 则数组元素中除存储二叉树结点的自身信息外, 还要存储其他的必要信息, 以反映二叉树的结构, 结点的存储次序可以是先根次序, 也可以是其他次序。

**【例 8-3】** 在查找算法中, 可用平均查找长度 (记为 ASL) 来衡量一个查找算法的优劣, 其定义为:

$$ASL = \sum_{i=1}^n p_i c_i$$

此处  $p_i$  为表中第  $i$  个记录被查找的概率,  $c_i$  为查到第  $i$  个记录时已进行的和关键字比较的次数,  $n$  为表中现有记录数。

以下叙述中均假定每一个记录被查找的概率相等, 即  $p_i = 1/n$ . ( $i = 1, 2, \dots, n$ ).





可选答案:

A、B、C、E: ① 7 ② 5 ③ 6 ④ 4 ⑤ 3

D: ① 27 ② 30 ③ 45 ④ 51 ⑤ 64

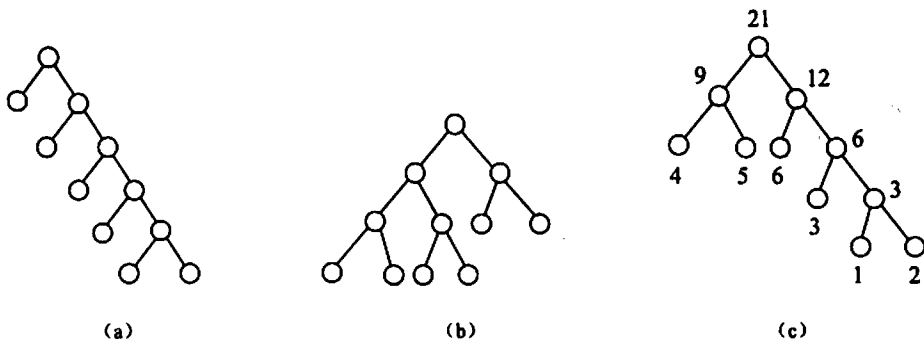
正确答案: A: ② B: ⑤ C: ② D: ④ E: ④

分析:

当将根结点的高度记为 0 时, 具有  $t$  个叶结点的正则二叉树的最大高度可达到  $t-1$ , 最小高度为  $\lceil \log_2 t \rceil$  (其中  $\lceil x \rceil$  表示不小于  $x$  的最小整数)。当  $t=6$  时, 最大高度为 5, 最小高度为  $\lceil \log_2 6 \rceil = 3$ , 在下图中, (a)所示的是 6 个叶结点的最高的正则二叉树。(b)所示的是 6 个叶结点的高度最小的正则二叉树, 在正则二叉树中, 非叶结点称为分支结点。设  $T$  中有  $i$  个分支结点,  $m$  条边,  $n$  个结点, 易知:  $m = 2i$ ,  $i+t=n$ ,  $m=n-1$ 。由此可推出  $i=t-1$ 。当  $t=6$  时,  $i=5$ 。

图 2.5(c)中所示的根树为带权为 1, 2, 3, 4, 5, 6 的最优二叉树 (利用 Huffman 算法得到), 树中非叶结点, 即分支结点所带权之和为  $3+6+12+9+21 = 51$ 。权为 1 的叶结点的所处层次为 4, 它处在最长通路上。

请注意, 有的书上规定根结点的所处层次为 1。若这样规定, 则 6 个叶结点的正则二叉树的最大高度为 6, 最小高度为 4, 在最优二叉树中, 权为 1 的叶结点所处层次为 5。



【例 8-5】从下列有关数据的存储结构的叙述中, 选出 5 条正确的叙述。

- (1) 顺序存储方式只能用于存储线性结构。
- (2) 顺序存储方式的优点是存储密度大, 且插入、删除运算效率高。
- (3) 链表的每个结点中都恰好包含一个指针。
- (4) 散列法存储的基本思想是由关键字的值决定数据的存储地址。
- (5) 散列表的结点中只包含数据元素自身的信息, 不包含任何指针。
- (6) 负载因子 (装填因子) 是散列法的一个重要参数, 它反映散列表的装满程度。
- (7) 栈和队列的存储方式既可是顺序方式, 也可是链接方式。

(8) 用二叉链表法 (lchild-rchild 法) 存储包含  $n$  个结点的二叉树, 结点的  $2n$  个指针区域中有  $n+1$  个为空指针。

(9) 用邻接矩阵法存储一个图时,在不考虑压缩存储的情况下,所占用的存储空间大只与图中顶点个数有关,而与图的边数无关。

(10) 邻接表法只能用于有向图的存储,而邻接矩阵法对于有向图和无向图的存储都适用。

正确答案: 正确的有 (4)、(6)、(7)、(8)、(9)

分析:

(1) 叙述不正确。一些非线性的结构也可以采用顺序方式存储。例如完全二叉树的存储,稀疏矩阵的三元组法存储,图的邻接矩阵法存储等。

(2) 叙述不正确。在顺序方式存储的数据结构中进行插入、删除运算,会引起大量结点的移动,因此运算效率不高。

(3) 叙述不正确。单链表的每个结点中包含一个指针,而双链表的每个结点中就包含两个指针。

(4) 叙述正确。散列法存储就是以结点的关键字  $k$  为自变量,通过散列函数  $h$  计算出函数值  $h(k)$ ,作为结点的存储地址。

(5) 叙述不正确。在散列法中,当不同的关键字值对应到同一存储地址,即  $k_1 \neq k_2$ ,但  $h(k_1) = h(k_2)$  时称做发生了冲突。若用拉链法处理冲突,就需要在散列表的每个结点中包括一个指针域,以指示对应到同一地址的下一个关键字值的实际存储地址。

(6) 叙述正确。散列表的负载因子

$$\alpha = \frac{\text{散列表中结点的数目}}{\text{散列表基本区域能容纳的结点的数目}}$$

它反映了散列表的装满程度,一般取  $\alpha < 1$ 。

(7) 叙述正确。虽然栈和队列通常用顺序方式存储,但它们完全可以用链接方式存储,而且在不少的的实际应用中采用的是链接方式存储(利用结点中已有的指针域)。

(8) 叙述正确。用二叉链表法(lchild-rchild)法存储包含  $n$  个结点的二叉树,每个结点有两个指针域,因此共有  $2n$  个指针域。除根结点外的  $n-1$  个结点都需要其双亲结点中的一个指针指向它,因此有  $n-1$  个非空指针,而其余的  $n+1$  个指针域中包含的都是空指针。

(9) 叙述正确。用邻接矩阵法存储图时,需要存储一个  $n \times n$  的矩阵,其中  $n$  是图中顶点个数。图中边的数目只影响矩阵中非零元素的个数,因此在不考虑压缩存储的情况下,占用的存储空间大小只与图中顶点个数有关,与边数无关。

(10) 叙述不正确。有向图和无向图都既可以用邻接矩阵法存储,又可以用邻接表法存储。无向图的邻接矩阵是对称矩阵。用邻接表法存储无向图,图的每一条边会在边表中出现两次。

【例 8-6】某顺序存储的表格,其中有 90000 个元素,已按关键字递增顺序排列。现假定对各个元素进行查找的概率相同,并且各个元素的关键字的值互不相同。

用顺序查找法查找时,平均比较次数约为 A, 最大比较次数为 B。

现把 90000 个元素按排列顺序划分成若干组, 使每组有  $g$  个元素 (最后一组可能不足  $g$  个)。查找时, 先从头一组开始, 通过比较各组的最后一个元素的关键字, 找到欲查找的元素所在的组, 然后再用顺序查找法找到欲查找的元素。在这种查找法中, 使总的平均比较次数最小的  $g$  是 C。此时的平均比较次数是 D。当  $g$  的值大于等于 90000 时。此方法的查找速度接近于 E。

可选答案:

- A、B: ① 25000                      ② 30000                      ③ 45000                      ④ 90000  
 C、D: ① 100                      ② 200                      ③ 300                      ④ 400  
 E: ① 快速排序                      ② 斐波那契查找                      ③ 二分法查找  
      ④ 顺序查找

正确答案: A: ③    B: ④    C: ③    D: ③    E: ④

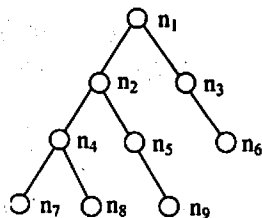
分析:

在包含  $n$  个元素的线性表中进行顺序查找, 平均比较次数为  $n/2$ , 最大比较次数为  $n$ 。题中的线性表包含 90000 个元素, 因此 A 处应填入选择③, 即 45000, B 处应填入选择④, 即 90000。

题中所述的第二种方法类似于分块查找。把 90000 个元素按排列顺序分组, 每组  $g$  个元素, 则组数为  $\lceil 90000 / g \rceil$ 。按题目所述的方法, 找到欲查找元素所在的组, 平均需要进行  $\lceil 90000 / g \rceil / 2$  次比较。组中有  $g$  个元素, 因此进一步找到欲查找的元素, 平均需要进行  $g/2$  次比较。总的平均比较次数为  $\lceil 90000 / g \rceil / 2 + g/2$ 。使总的平均比较次数最小的  $g$  为 300, 这时总的平均比较次数也为 300。因此 C 处和 D 处都应填入选择③, 即 300。当  $g$  的值大于等于 90000 时, 整个表只分为一个组, 实际上是对整个表进行顺序查找, 因此 E 处应填入选择④, 即顺序查找法。

【例 8-7】考虑具有如下性质的二叉树: 除叶结点外, 每个结点的值都大于其左子树上的一切结点的值, 并小于等于其右子树上的一切结点的值。

现把 9 个数 1, 2, 3, ..., 8, 9 填入右图所示的二叉树的 9 个结点中, 并使之具有上述性质。此时,  $n_1$  的值是 A,  $n_2$  的值是 B,  $n_9$  的值是 C。现欲把  $\sqrt{10}$  放入此树并使该树保持前述性质, 增加一个结点可以放在 D 或 E。



可选答案:

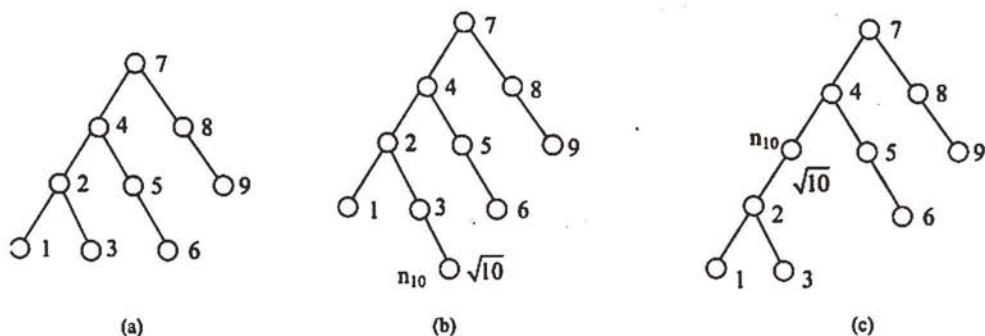
- A~C: ① 1                      ② 2                      ③ 3  
      ④ 4                      ⑤ 5                      ⑥ 6  
      ⑦ 7                      ⑧ 8                      ⑨ 9  
 D、E: ①  $n_7$  下面                      ②  $n_8$  下面                      ③  $n_9$  下面  
      ④  $n_6$  下面                      ⑤  $n_1$  与  $n_2$  之间                      ⑥  $n_2$  与  $n_4$  之间  
      ⑦  $n_5$  与  $n_9$  之间                      ⑧  $n_3$  与  $n_6$  之间

正确答案: A: ⑦ B: ④ C: ⑥ D: ② E: ⑥

分析:

将 1, 2, 3, ..., 8, 9, 填入题图所示二叉树的各结点, 使其成为下图(a)图的形式。显然, 在此二叉树中, 各结点的值具有所要求的性质。由此可知,  $n_1$  的值为 7,  $n_2$  的值为 4,  $n_9$  的值为 6。

由于  $3 < \sqrt{10} < 4$ , 将原二叉树增加一个结点, 使其值为  $\sqrt{10}$ , 并保证新二叉树各结点之值仍具有所要求的性质, 有两种方案。其一是在  $n_8$  的下方增加一个结点  $n_{10}$ , 使其成为  $n_8$  的右子女, 其值为  $\sqrt{10}$ , 见图 2 中(b)所示。其二是在  $n_2$  与  $n_4$  之间增加一个结点  $n_{10}$ , 使得  $n_{10}$  为  $n_2$  的左儿子,  $n_4$  为  $n_{10}$  的左子女,  $n_{10}$  的值为  $\sqrt{10}$ , 见图 2 中(c)图所示。根据(b), (c)图, D 和 E 的位置就显而易见了。



【例 8-8】 从下列叙述中选出 5 条正确的叙述。

- (1)  $m$  阶 B-树每一个结点的子树棵数都小于等于  $m$ 。
- (2)  $m$  阶 B-树每一个结点的子树棵数都大于等于  $\lceil m/2 \rceil$ 。
- (3)  $m$  阶 B-树具有  $k$  个子树的非失败结点含有  $k-1$  个关键字。
- (4)  $m$  阶 B-树的任何一个结点的左右子树的高度都相等。
- (5) 中序遍历一棵二叉查找树的结点就可得到排好序的结点序列。
- (6) 用指针的方式存储一棵有  $n$  个结点的二叉树, 最少要  $n+1$  个指针。
- (7) 任一查找树的平均查找时间都小于顺序查找法查找同样结点的线性表的平均查找时间。
- (8) 平衡树一定是丰满树。
- (9) 已知树的前序遍历并不能惟一地确定这棵树, 因为不知道树的根结点是哪一个。
- (10) 不使用递归, 也可实现二叉树的前序、中序及后序遍历。

正确答案: 正确的有(1)、(3)、(5)、(6)、(10)

分析:

- (1) 叙述正确。根据定义, 一棵  $m$  阶 B-树的每一个结点最多可有  $m$  棵子树。
- (2) 叙述不正确, 因为根结点的子树棵数可以少于  $\lceil m/2 \rceil$  个, 正确的叙述应是:



除根结点之外的所有非失败结点至少有  $\lceil m/2 \rceil$  棵子树。

(3) 叙述正确。根据定义, 一棵  $m$  阶  $B$ -树的非失败结点若具有  $k$  个子树, 则应有  $k-1$  个关键字。

(4) 叙述不正确:  $m$  阶  $B$ -树并不一定是二叉树, 故不能有“左右子树”之说。

(5) 叙述正确。

(6) 叙述正确。

(7) 叙述正确, 含有  $n$  个结点的二叉查找树的平均查找时间和树的形态有关, 当这几个结点的记录已按关键字有序时, 构成的二叉查找树就蜕变为单支树, 树的深度为  $n-1$ , 其平均查找长度为  $(n+1)/2$  和顺序查找相同, 这是最坏的情况。

(8) 叙述不正确, 平衡树不一定是丰满树, 而丰满树则一定是平衡树。因为平衡树中的结点的左右子树的高度允许相差 1。

(9) 叙述是正确。虽然前半句话正确, 但后面树的根结点是知道的。

(10) 叙述正确。

【例 8-9】数据结构中, 与所使用的计算机无关的是数据的 A 结构; 链表是一种采用 B 存储结构存储的线性表, 链表适用于 C 查找, 但在链表中进行 D 操作的效率比在顺序存储结构中进行 D 操作的效率高; 二分法查找 E 存储结构。

可选答案:

- |             |          |                |                  |
|-------------|----------|----------------|------------------|
| A: ① 存储     | ② 物理     | ③ 逻辑           | ④ 物理和存储          |
| B: ① 顺序     | ② 链式     | ③ 星式           | ④ 网状             |
| C: ① 顺序     | ② 二分法    | ③ 顺序, 也能二分法    | ④ 随机             |
| D: ① 顺序查找   | ② 二分法查找  | ③ 快速查找         | ④ 插入             |
| E: ① 只适合于顺序 | ② 只适合于链式 | ③ 既适合于顺序也适合于链式 | ④ 既不适合于顺序也不适合于链式 |

正确答案: A: ③ B: ② C: ① D: ④ E: ①

分析:

数据结构概念一般包括三个方面的内容: 数据的逻辑结构、存储结构以及数据上的运算集合。

数据的逻辑结构只抽象地反映数据元素之间的逻辑关系, 而不管它在计算机中的存储表示形式。数据的逻辑结构分为线性结构和非线性结构, 若各数据元素之间的逻辑关系可以用一个线性序列简单地表示出来, 则称之为线性结构, 线性表是典型的线性结构, 而树、图等都是非线性结构。

数据的存储结构是逻辑结构在计算机存储器里的实现, 数据的存储方式主要有顺序存储结构和链式存储结构, 顺序存储结构主要用于线性的数据结构, 它把逻辑上相邻的数据元素存储在物理上相邻的存储单元的, 结点之间的关系由存储单元的邻接关系来实现, 而链式存储结构是在每个结点中至少包括一个指针域, 用指针来体现数据元素

之间逻辑上的联系。

链表就是链式存储的线性表，链表的一个重要特点是插入、删除运算灵活方便，不需移动结点，只要改变结点中指针域的值便可，顺序存储结构插入、删除运算不便，会引起大量结点的移动。

查找是数据结构中的常用运算，通常的方法是顺序查找和二分法查找。顺序查找是线性表的最简单的查找方法，它用待查关键字值与线性表中各个结点的关键字值逐个比较，直到找到相等的关键字值，其特点是对线性表的顺序存储结构和链式存储结构均适用。二分法查找是一种效率较高的线性表查找方法，它用待查关键字值与线性表中间位置结点的关键字值相比较，这个中间结点把线性表分成了两个子表，比较相等则查找完成，否则根据比较结果确定下一步的查找应在哪一个子表中进行，如此往复，直到找到满足条件的结点。进行二分法查找的先决条件是线性表结点必须是按关键字值排好序的，且线性表以顺序方式存储，所以链表不适于二分法查找。

【例 8-10】 从供选择的答案中选出与下列各术语关系最密切的答案，把编号与术语对应起来。

- A 后进先出                      B 先进先出                      C 高级程序设计语言  
D 分时系统                      E 输入输出处理

可选答案：

- A~E: ① 时间片                      ② 存储保护   ③ 数据类型                      ④ 栈  
⑤ 高速缓冲存储器   ⑥ 通道                      ⑦ 基址寄存器                      ⑧ 变址寄存器  
⑨ 作业流                      ⑩ 队列

正确答案：A: ④   B: ⑩   C: ③   D: ①   E: ⑥

分析：

(1) 栈是一种特殊的、限定仅在表的一端进行插入和删除运算的线性表，这一端称为栈顶，另一端则叫做栈底。若栈中有从栈底到栈顶的元素顺次为  $a_1, a_2, \dots, a_n$  等元素，新元素进栈要置于  $a_n$  之上；删除或退栈必须对  $a_n$  进行，这就形成了“后进先出”操作原则，因此 A 对应于④。

(2) 队列是一种特殊的、限定所有的插入都在表的一端进行、所有的删除都在表的另一端进行的线性表，进行删除的一端叫队列的头，进行插入的一端叫队列的尾。在队列中，新元素总是加入到队尾。每次删除总是在队列头上的，这就形成了先进先出的操作原则，因此 B 对应于⑩。

(3) 任何一种程序设计语言都可以看成是对一些数据以及按某种次序作用于该组数据上操作的一种说明，不同的语言所提供的数据类型不尽相同，因此 C 对应于③。

(4) 分时系统允许多个用户同时联机地与计算机系统进行交互通信。分时系统采用时间片轮转的方法来处理用户的服务请求，即规定每个用户一次可以使用 CPU 的时间（称为时间片），按某个轮换次序在用户之间分配处理机，因此 D 对应于①。

(5) 通道是独立于 CPU 的专用处理机, 用来管理输入输出工作, 它对外设实现统一管理, 代替 CPU 对输入输出操作实施控制, 因此 E 对应于⑥。

【例 8-11】 散列法存储的基本思想是根据 A 来决定 B, 冲突指的是 C, D 越大, 发生冲突的可能性也越大。处理冲突的两类主要方法是 E。

可选答案:

A、B、D: ① 存储地址 ② 元素的序号 ③ 元素个数 ④ 关键字值  
⑤ 非码属性 ⑥ 平均查找长度 ⑦ 负载因子 ⑧ 散列表空间

C: ① 两个元素具有相同序号  
② 两个元素的关键字值不同, 而非关键字相同  
③ 不同关键字值对应到相同的存储地址  
④ 装载因子过大  
⑤ 数据元素过多

E: ① 线性探查法和双散列函数法  
② 建溢出区法和不建溢出区法  
③ 质数除取余法和折叠法  
④ 拉链法和开地址法

正确答案: A: ④ B: ① C: ③ D: ⑦ E: ④

分析:

散列表是线性表的一种重要存储和查找方法, 在散列表里可以对结点进行快速查找。散列法存储的基本思想是: 由结点的关键字值决定结点的存储地址, 即以关键字  $k$  为自变量, 通过一定的函数  $h$  (称为散列函数), 计算出对应的函数值  $h(k)$ , 把这个值解释为结点的存储地址。

在散列法中, 不同的关键字对应到同一存储地址, 即  $k_1 \neq k_2$  但  $H(k_1) = H(k_2)$  的现象称为冲突; 发生冲突的两个或多个关键字值称为同义词。

散列表的一个重要参数是装载因子  $\alpha$ , 它定义为

$$\alpha = \frac{\text{散列表中结点的数目}}{\text{基本区域能容纳的结点数目}}$$

装载因子的大小体现散列表的装满程度,  $\alpha$  越大, 则散列表装得越满, 发生冲突的可能性越大, 一般取  $\alpha < 1$ 。

处理冲突的方法主要有两种: 拉链法和开地址法。

用拉链法处理冲突就是给某列表的每个结点增加一个 link 字段, 当冲突发生时利用 link 字段拉链, 建立链接方式的同义词子表。每个同义词子表的第一个元素都在某列表基本区域中, 同义词子表的其他元素存储在何处, 又有两种解决方法, 一种是建立溢出区, 存放各种同义词子表的其他元素; 另一种是不建溢出区, 同义词子表的其他元素就存放在基本区域中没有占用的单元中。



用开地址法处理冲突就是当冲突发生时形成一个探查序列, 沿着这序列逐个地址探查, 直至找到一个开放的地址(未被占用的单元), 将发生冲突的关键字值存入该地址中。最简单的探查序列是线性探查, 即若发生冲突的地址为  $d$ , 则探查的地址序列为

$$d+1, d+2, \dots, m-1, 0, 1, \dots, d-1$$

其中  $m$  为散列表存储区域的大小。

另一种效果更好的探查序列是双散列函数法, 即用第二个散列函数  $h_2$  来确定探查序列, 若发生冲突的地址为  $d$ , 则探查的地址序列为

$$(d+h_2(k)) \% m, (d+2h_2(k)) \% m, (d+3h_2(k)) \% m, \dots$$

除余法和折叠法是两种常用的散列函数法。除余法是选择一个适当的正整数  $p$  (通常选  $p$  为小于散列表存储区域大小的最大质数), 用  $p$  去除关键字值, 取其余数为地址。折叠法是将关键字的值从某些地方断开, 分为几段, 折叠相加, 作为地址。

**【例 8-12】** 在内排序的过程中, 通常需要对待排序的关键字集合进行多遍扫描。采用不同排序方法, 会产生不同的排序中间结果。设要将序列 (Q, H, C, Y, P, A, M, S, R, D, F, X) 中的关键字按字母序的升序重新排列, 则 A 是冒泡排序一趟扫描的结果, B 是初始步长为 4 的希尔 (Shell) 排序一趟扫描的结果, C 是二路归并 (合并) 排序一趟扫描的结果, D 是以第一个元素为基准元素的快速排序一趟扫描的结果, E 是堆排序初始建堆的结果。

可选答案:

- A~E: ① F, H, C, D, P, A, M, Q, R, S, Y, X  
 ② P, A, C, S, Q, D, F, X, R, H, M, Y  
 ③ A, D, C, R, F, Q, M, S, Y, P, H, X  
 ④ H, C, Q, P, A, M, S, R, D, F, X, Y  
 ⑤ H, Q, C, Y, A, P, M, S, D, R, F, X

正确答案: A: ④ B: ② C: ⑤ D: ① E: ③

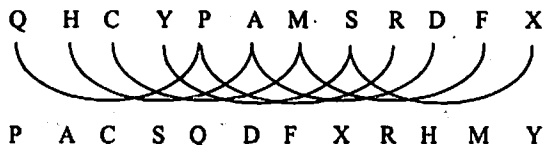
分析:

排序是数据处理中经常使用的一种重要运算, 设  $(R_0, R_1, \dots, R_{n-1})$  是由  $n$  个记录组成的文件, 其相应的关键字序列为  $(K_0, K_1, \dots, K_{n-1})$ , 所谓排序就是将记录按其关键字值不增 (或不减) 的次序排列起来, 所谓内排序是指整个排序过程都在内存进行的排序。

冒泡排序是将待排序的记录顺次两两比较, 若为逆序则进行交换。将序列照此方法从头到尾处理一遍的效果是将关键字值最大的记录交换到了最后的位置。

希尔排序是按增量将文件分组。首先取增量  $d_1 < n$ , 把全部记录分成  $d_1$  个组, 所有距离为  $d_1$  倍数的记录放在一组中, 各组内用直接插入排序法排序; 然后取  $d_2 < d_1$ , 重复上述分组和排序工作, 直至取  $d_i = 1$ , 即所有记录放在一个组中排序为止, 其一趟扫描的过程和结果如下图所示:



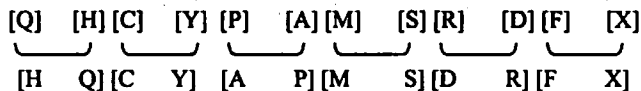


二路归并排序是将两个已排序的文件合并使之成为第三个已排序的文件,其步骤如下:

(1) 首先把  $n$  个记录看成  $n$  个长度为 1 的已排序文件,把这些文件成对地加以合并,得到长度为 2 的  $[(n-1)/2]+1$  个已排序的文件 ( $[x]$  表示不大于  $x$  的最大整数)。

(2) 然后再将这  $[(n-1)/2]+1$  个文件成对地合并,以此类推,直到最后得到一个长度为  $n$  的文件为止。

该方法一趟扫描的过程和结果如下图所示:



快速排序的步骤是:

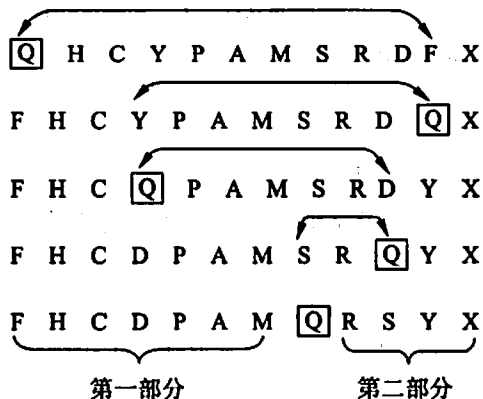
(1) 在待排序序列中任取一个记录,以它为基准用交换的方法将所有的记录分成两部分,关键字值比它小的在一部分,关键字值比它大的在另一部分。

(2) 分别对这两个部分实施上述过程,一直重复到排序完成。

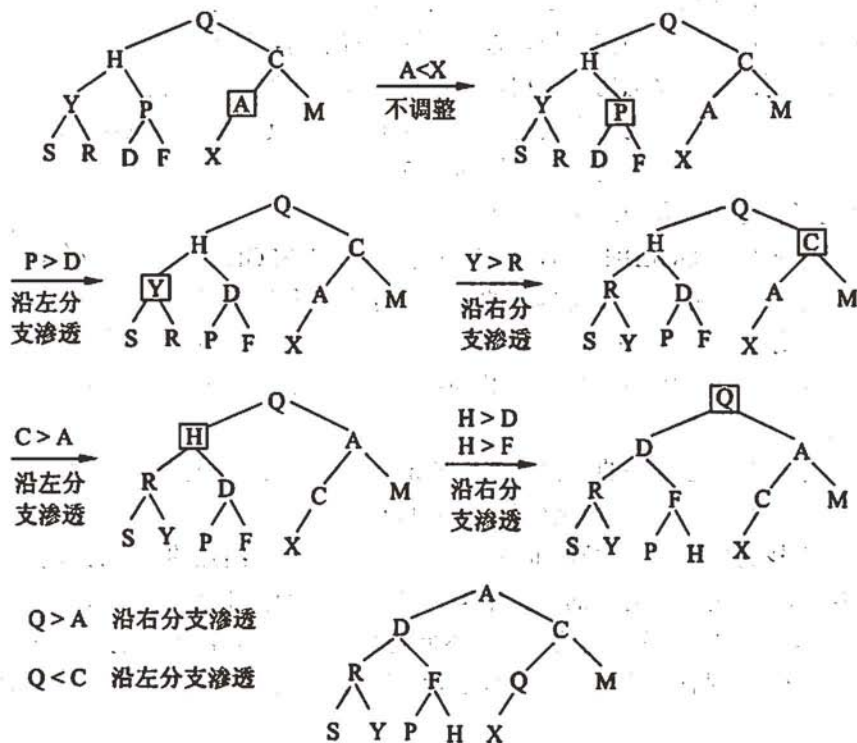
该方法一趟扫描的过程是取第一个元素  $Q$  作为将所有

[Q H C Y P A M S R D F X]

记录分成两部分的基准,将比  $Q$  大的关键字移到后面,将比  $Q$  小的关键字移到前面,为节省空间,移动方法采用从两端往中间夹入的方式。即先取出,这样空出前端第一个关键字的位置,用  $Q$  和  $X$  比较,  $X$  留在原处不动,继续用  $Q$  和  $F$  比较,  $F$  移到原来  $Q$  的位置,从而空出  $F$  的位置,这时  $Q$  和  $H$  比较,再和  $C$  比较,如此往复比较、交换,一步步地往中间夹入,最后在空出的位置上填入  $Q$ ,便完成了一趟排序,该过程和结果如下图所示。



堆排序初始建堆是首先将要排序的所有关键字放到一棵完全二叉树的各个结点中，然后从  $i = [(n-1)/2]$ ，即最后一个非叶结点  $k_i$  开始，逐步把以  $K_{[(n-1)/2]}$ ， $K_{[(n-1)/2]-1}$ ， $K_{[(n-1)/2]-2}$ ，... 为根的子树排成堆，直到以  $K_0$  为根的树排成堆便完成了建堆过程。这里堆的特性在完全二叉树里解释为：完全二叉树中任一结点的关键字值都小于或等于它的子女结点的关键字，所以根结点  $K_0$  是完全二叉树中关键字值最小的结点。初始建堆的过程如下图所示。



### 【例 8-13】 单选填空

1. 已知一棵二叉树的前序序列和中序序列分别为：ABDEGCFH 和 DBGEACHF，则该二叉树的后序序列为 A，层次序列为 B。
2. 设有  $n$  个结点进行排序，不稳定排序是 C；快速排序的最大比较次数是 D。
3. 设有 100 个结点，用二分法查找时，最大比较次数是 E。

可选答案：

A、B: ① GEDHFBCA

② DGEHBHFC

③ ABCDEFGH

④ ACBFEDHG

C: ① 直接插入排序

② 冒泡排序

③ Shell 排序

④ 归并排序

D: ①  $n \log_2 n$

②  $n^2$

③  $n^2/2$

④  $n$

E: ① 25                      ② 50                      ③ 10                      ④ 7

正确答案: A: ②    B: ③    C: ③    D: ②    E: ④

分析:

遍历是二叉树的一种重要运算。遍历一个二叉树就是按一定的次序系统地访问该结构中的所有结点,使每个结点恰被访问一次。

可以按多种不同的次序遍历树形结构,常用的有以下几种。

前序遍历:访问根,按前序遍历左子树,按前序遍历右子树。

后序遍历:按后序遍历左子树,按后序遍历右子树,访问根。

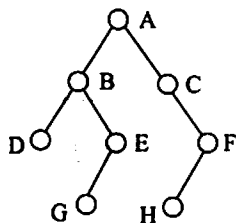
中序法遍历:按中序遍历左子树,访问根,按后序遍历右子树。

层次序遍历:按广度优先方式遍历,即首先依次访问层数为0的结点,然后依次访问层数为1的结点,直至访问最下一层的所有结点。

由以上概念可知,题中二叉树如下:

所以后序序列为 DGEHBFC A,层次序列为 ABCDEFGH。

所谓稳定排序是指:如果待排序的文件中存在多个具有相同关键字的记录,经过排序后这些记录的相对次序仍然保持不变,则这种排序算法称为稳定的,反之称为不稳定的。Shell 排序是一种不稳定排序算法。



快速排序是对冒泡排序的一种改进。对  $n$  个记录的文件进行快速排序,在最坏的情况下执行时间为  $O(n^2)$ ,其平均执行时间  $O(n \log_2 n)$ 。

二分法查找的最大比较次数  $\lceil \log_2 n \rceil$ ,所以查找 100 个结点的最大比较次数是 7。

【例 8-14】堆是一种有用的数据结构。例如关键字序列 A 是一个堆。

堆排序是一种 B 排序,它的一个基本问题是如何建堆,常用的建堆算法是 1964 年 Floyd 提出的 C。对含  $n$  个元素的序列进行排序时,堆排序的时间复杂性是 D,所附加存储结点是 E。

可选答案:

A: ① 16, 72, 31, 23, 94, 53                      ② 94, 53, 31, 72, 16, 53

③ 16, 53, 23, 94, 31, 72                      ④ 16, 31, 23, 94, 53, 72

⑤ 94, 31, 53, 23, 16, 72

B: ① 插入                      ② 选择                      ③ 交换                      ④ 基数                      ⑤ 归并

C: ① 淘汰法                      ② 渗透法                      ③ 递推法                      ④ LRU 算法

D、E: ①  $O(n \log_2 n)$     ②  $O(n)$                       ③  $O(\log_2 n)$     ④  $O(n^2)$                       ⑤  $O(1)$

正确答案: A: ④    B: ②    C: ②    D: ①    E: ⑥

分析:

堆是一个关键字序列  $(K_0, K_1, K_2, \dots, K_{n-1})$ ,它具有如下特性:  $K_i \leq K_{2i+1}, K_i \leq K_{2i+2}$ , 这里  $i = 0, 1, 2, \dots, [(n-1)/2]$ 。

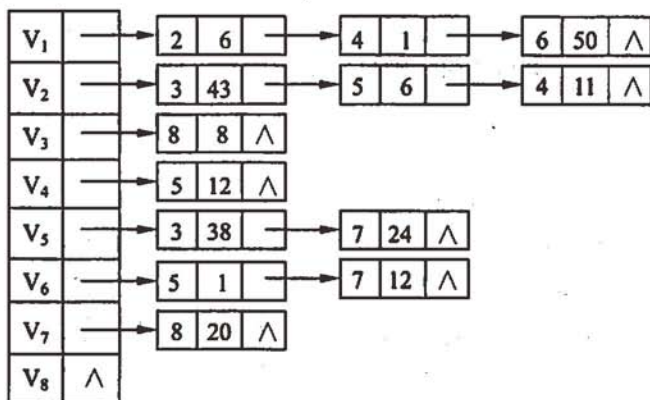
堆实质上是一棵完全二叉树的层次排列, 此完全二叉树的每一个结点对应于一个关键字, 根结点对应于  $K_0$ 。堆的特性在此完全二叉树中的解释为: 完全二叉树中任一结点的关键字值都小于或等于它的两个子女结点的关键字值。由此可知, (16, 31, 23, 94, 53, 72) 是一个堆。

堆排序是一种选择排序。选择排序的基本思想是: 每次从待排序的记录中选择出关键字最小(或最大)的记录, 顺序放在已排序的记录序列的最后, 直到全部排完。

堆排序的基本问题是如何建堆, 常用要建堆算法称为渗透法。

堆排序的执行时间是  $O(n\log_2 n)$ , 且仅需要一个用于交换的附加存储结点, 因此是一种适合于对较大文件进行排序的方法。

**【例 8-15】** 下图是带权的有向图  $G$  的邻接表表示法。从结点  $V_1$  出发深度遍历图  $G$  所得的结点序列为 A; 广度遍历图  $G$  所得的结点序列是 B;  $G$  的一个拓扑序列是 C; 从结点  $V_1$  到结点  $V_8$  的最短路径是 D; 从结点  $V_1$  到  $V_8$  的关键路径是 E。



可选答案:

- A~C: ①  $V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8$       ②  $V_1, V_2, V_4, V_6, V_5, V_3, V_7, V_8$   
 ③  $V_1, V_2, V_4, V_6, V_3, V_5, V_7, V_8$       ④  $V_1, V_2, V_4, V_6, V_7, V_3, V_5, V_8$   
 ⑤  $V_1, V_2, V_3, V_8, V_4, V_5, V_6, V_7$       ⑥  $V_1, V_2, V_3, V_8, V_4, V_5, V_7, V_6$   
 ⑦  $V_1, V_2, V_3, V_8, V_5, V_7, V_4, V_6$   
 D、E: ①  $(V_1, V_2, V_4, V_5, V_3, V_8)$       ②  $(V_1, V_6, V_5, V_3, V_8)$   
 ③  $(V_1, V_6, V_7, V_8)$       ④  $(V_1, V_2, V_5, V_7, V_8)$

正确答案: A: ⑦    B: ③    C: ②    D: ④    E: ②

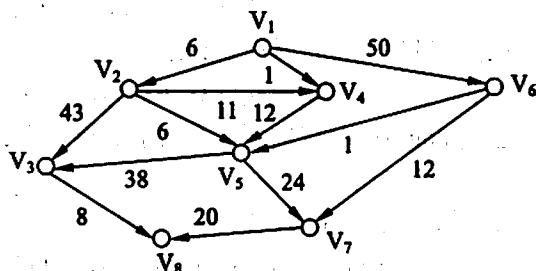
分析:

图的存储方式多种多样, 邻接表表示法是常用的一种, 它保存一个顺序存储的顶点表和  $n$  个链接存储的边链表。顶点表的每一个表目对应于图的一个顶点, 每个表目包括两个字段: 一个是顶点的数据或指向顶点数据的指针, 另一个是指向此顶点的边链表的



指针。图的每个顶点都有一个边链表，一个顶点的边链表的每个链结点对应于与该顶点相关联的一条边，它也包括两个字段：一个是与此边相关联的另一个顶点的顶点序号；另一个是指向边链表下一个边结点的指针。

因此，题中邻接表所对应的图如下所示：



图的深度优先遍历是从图中某个结点  $V_1$  出发，访问此结点，然后依次从  $V_1$  的未被访问的邻接顶点出发进行深度优先遍历，直至图中所有和  $V_1$  有路径相通的结点都被访问到。若此时图中尚有顶点未被访问，则另选图中一个未被访问过的顶点作起始顶点，重复上述过程，直至图中所有顶点都被访问到为止。因此，图  $G$  的深度优先遍历是  $V_1, V_2, V_3, V_8, V_5, V_7, V_4, V_6$ 。

广度优先遍历是先访问顶点  $V_1$ ，然后访问  $V_1$  邻接到的所有未被访问过的顶点  $V_2, V_3, \dots, V_i$ ，再依次访问  $V_2, V_3, \dots, V_i$  邻接到的所有未被访问的顶点。如此进行下去，直到访问遍所有顶点，因此图  $G$  的广度优先遍历是  $V_1, V_2, V_4, V_6, V_3, V_5, V_7, V_8$ 。

有向图的顶点的线性序列  $(V_{i1}, V_{i2}, \dots, V_{in})$  称做一个拓扑序列，如果该顶点序列满足如下条件：若从顶点  $V_i$  到  $V_j$  有一条路径，则在序列中顶点  $V_i$  必定在  $V_j$  之前，任何无环有向图的顶点都可以排在一个拓扑序列中。但是，拓扑序列不是惟一的。拓扑排序的方法是反复执行如下两步：

- (1) 从图中选择一个入度为 0 的顶点且输出之；
- (2) 从图中删除此顶点及其所有的出边。

当所有的顶点都输出了，拓扑排序即告完成。在可供选择的答案中，②是一个拓扑序列。

最短路径是两个顶点间长度最短的路径。这里的路径长度是指路径上的边所带权的总和，而不是路径上边数的总和。关键路径是指两个顶点间具有最大长度的路径。通过计算可知， $(V_1, V_2, V_5, V_7, V_8)$  是最短路径， $(V_1, V_6, V_5, V_3, V_8)$  是关键路径。

【例 8-16】 设数据结构  $(D, R)$  由数据结点集合  $D = \{d_i \mid 1 \leq i \leq 7\}$  及其上的关系  $R$  组成。

- (1) 当  $R = \{<d_{i-1}, d_i> \mid d_{i-1}, d_i \in D, 2 \leq i \leq 7\}$ ，这个数据结构对应于 A。
- (2) 当  $R = \{<d_4, d_2>, <d_2, d_1>, <d_2, d_3>, <d_4, d_6>, <d_6, d_5>, <d_6, d_7>\}$ ，这个结构的

图形是 B；用 C 遍历法可以得到 A 的数据结构。

(3) 当  $R = \{ \langle d_1, d_2 \rangle, \langle d_1, d_3 \rangle, \langle d_2, d_4 \rangle, \langle d_3, d_4 \rangle, \langle d_4, d_5 \rangle, \langle d_4, d_6 \rangle, \langle d_4, d_7 \rangle \}$ ，这个结构的图形是 D；用 E 遍历法可以得到 A 的数据结构。

可选答案：

- A, B, D:    ① 二叉树                  ② 队列                  ③ 二叉查找树  
                  ④ 线性表                ⑤ 无向图                ⑥ 有向无回路图  
 C, E:        ① 前序                  ② 中序                  ③ 后序  
                  ④ 深度优先            ⑤ 广度优先

正确答案：A: ④    B: ①    C: ②    D: ⑥    E: ⑤

分析：

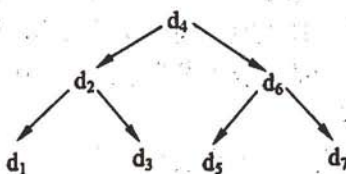
数据结构  $(D, R)$  中， $D$  由 7 个数据组成， $R$  决定这 7 个数据的排列方式，下面逐个分析不同的  $R$  对应的不同数据结构：

①  $R = \{ \langle d_{i-1}, d_i \rangle \mid d_{i-1}, d_i \in D, 2 \leq i \leq 7 \}$ ，对应的数据结构可以表示为：

$d_1 \rightarrow d_2 \rightarrow d_3 \rightarrow d_4 \rightarrow d_5 \rightarrow d_6 \rightarrow d_7$

因此它对应一个线性表。

②  $R = \{ \langle d_4, d_2 \rangle, \langle d_2, d_1 \rangle, \langle d_2, d_3 \rangle, \langle d_4, d_6 \rangle, \langle d_6, d_5 \rangle, \langle d_6, d_7 \rangle \}$ ，对应的数据结构可以表示为：



可以看出，这是一棵二叉树，利用中序遍历法得到序列为  $d_1 d_2 d_3 d_4 d_5 d_6 d_7$ ，正好对应①中的数据结构。

③  $R = \{ \langle d_1, d_2 \rangle, \langle d_1, d_3 \rangle, \langle d_2, d_4 \rangle, \langle d_3, d_4 \rangle, \langle d_4, d_5 \rangle, \langle d_4, d_6 \rangle, \langle d_4, d_7 \rangle \}$ ，对应的数据结构可以表示为：

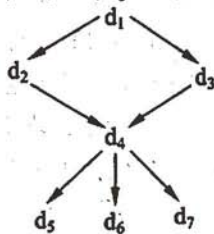
这可以对应一个有向无回路图，利用广度优先遍历法得到的序列为： $d_1 d_2 d_3 d_4 d_5 d_6 d_7$ ，正好对应 ① 中的线性表结构。

该题要求考生对数据结构中常用的线性表、二叉树、树、图的结构及其遍历方法应该熟练掌握。

【例 8-17】在下列程序中：

```

void calc(int p1, int p2){
    p2=p2*p2; p1=p1-p2; p2=p2-p1;
} /*calc*/
  
```



```
void main{
    int i=2, j=3;
    calc(i,j); cout<<j<<endl;
} /*main*/
```

当参数传递改用引用方式 (Call by reference) 时, 所得结果  $j = \underline{A}$ ;

当参数传递改用换名方式 (Call by name) 时, 所得结果  $j = \underline{B}$ ;

当参数传递采用赋值方式 (Call by value) 时, 所得结果  $j = \underline{C}$ 。

递归是程序设计中一种重要的控制结构, 通常实现递归时, 采用的数据结构是 D。

对那些既可以用递归方式, 也可以用循环方式求解的问题, 就执行效率而言 E。

可选答案:

A~C: ① 0                      ② 3                      ③ 5                      ④ 6  
          ⑤ 10                      ⑥ 16                      ⑦ 20                      ⑧ 28

D: ① 数组                      ② 栈                      ③ 队列                      ④ 循环链表

E: ① 难以断定                      ② 两者相同                      ③ 循环优于递归                      ④ 递归优于循环

正确答案: A: ⑥    B: ⑥    C: ②    D: ②    E: ③

分析:

引用方法传递参数的实质是传送地址, 所以子程序中的操作将影响变量本身的内容。

换名方式传递参数是给同一地址赋不同的变量名称, 修改其中一个变量的内容, 另一个变量的内容也发生了改变, 所以子程序的操作将影响变量本身内容。

赋值方式传递参数只是将变量的内容赋给参数, 子程序中对参数的操作不会影响变量本身的内容。

递归采用的数据结构是栈, 对于既可以用递归实现, 又可以用循环方式求解的问题, 就执行效率而言, 循环优于递归, 因为递归开销大, 每次调用都要入栈、出栈, 保存现场, 恢复现场。

【例 8-18】一棵二叉查找树可顺序存放在一组物理上相邻的存储区中, 每个结点及其左右指针依次分别存放在该存储区的三个连续单元中。现对一棵按结点字母的字典顺序购成的二叉查找树, 从根结点 P 开始顺序存放在一个存储区中, 结果如图所示, 其中  $L_i$  为第  $i$  个结点的左指针,  $R_i$  为第  $i$  个结点的右指针, 则  $L_2$  应为 A,  $L_4$  应为 B,  $R_1$  应为 C。该二叉查找树的前序遍历序列为 D, 后序遍历序列为 E。

可选答案:

A~C: ① 1003                      ② 1004                      ③ 100A

1000	P
1001	$L_1$
1002	$R_1$
1003	B
1004	$L_2$
1005	$R_2$
1006	Q
1007	$L_3$
1008	$R_3$
1009	H
100A	$L_4$
100B	$R_4$
100C	C
100D	$L_5$
100E	$R_5$
100F	J
1010	$L_6$
1011	$R_6$

④ 1009

⑤ 1006

⑥ 1000

⑦ 100C

⑧ 100F

⑨ NULL

D、E: ① PBQHCJ

② PBHCJQ

③ BCHJPQ

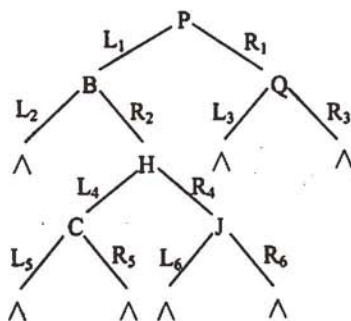
④ CJHBQP

⑤ BHCJQP

正确答案: A: ⑨ B: ⑦ C: ⑤ D: ② E: ④

分析:

根据二叉查找树的特点: 每个结点的左子树中所有的结点的关键字值都小于该结点的关键字值, 而右子树中所有结点的关键字值都大于该结点的关键字值, 则可知由根结点 P 开始, 按结点字母的字典顺序构成的二叉查找树为:



显然,  $L_2$  指向 NULL,  $L_4$  指向 C, 即 100C,  $R_1$  指向 Q, 即 1006, 其前序遍历序列为 PBHCJQ, 后序遍历序列为 CJHBQP。

【例 8-19】对由  $n$  个记录所组成的表按关键字排序时, 下列各常用排序算法的平均比较次数分别是: 二路归并排序为 A, 冒泡排序为 B, 快速排序为 C。其中, 归并排序和快速排序所需要的辅助存储分别是 D 和 E。

可选答案:

A~C: ①  $O(1)$ ②  $O(n \log_2 n)$ ③  $O(n)$ ④  $O(n^2)$ ⑤  $O(n(\log_2 n)^2)$ ⑥  $O(\log_2 n)$ 

正确答案: A: ② B: ④ C: ② D: ③ E: ⑥

分析:

二路归并排序是将初始序列看成  $n$  个有序的子序列, 每个子序列的长度为 1, 然后两两合并, 得到  $\lceil n/2 \rceil$  个长度为 2 或 1 的有序子序列; 再两两合并, …… , 如此重复, 直至得到一个长度为  $n$  的有序序列为止。二路归并排序的时间复杂度为  $O(n \log_2 n)$ 。

冒泡排序是一种利用交换进行排序的方法, 其过程为: 每一趟选出一个最大(最小)的元素, 放在未排序序列的最后位置上, 然后在剩下的  $n-1$  个元素的序列上进行同样操作, 直到剩下一个元素为止, 它总的时间复杂度为  $O(n^2)$ 。



快速排序是对冒泡排序的一种改进,它通过一趟排序将待排记录分割成独立的两部分,其中一部分记录的关键字均比另一部分记录的关键字小,则可分别对这两部分记录继续进行排序,以达到整个序列有序,快速排序的平均时间复杂度为  $O(n\log_2 n)$ 。

归并排序需要有和待排记录等数量的存储空间,因而为  $O(n)$ ,快速排序需要的辅助存储为  $O(\log_2 n)$ 。

【例 8-20】在排序算法中,两两比较待排序的记录,当发现不满足顺序要求时,变更它们的相对位置,这就是 A 排序。每次从未排序的记录中挑出最小(或最大)关键字值的记录,加入到已排序记录的末尾,这是 B 排序。堆排序是一种 C 排序,堆是一种数据结构,如关键字序列 D 就组成一个堆,堆排序的平均执行时间和需附加的存储结点分别为 E。

可选答案:

A~C: ① 插入                  ② 枚举                  ③ 交换                  ④ 归并  
         ⑤ 基数                  ⑥ 选择                  ⑦ 希尔

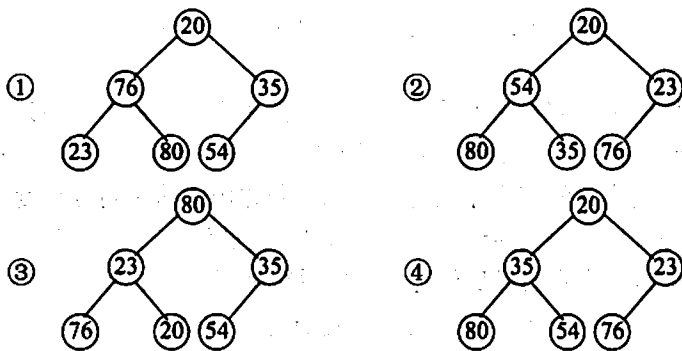
D: ① 20, 76, 35, 23, 80, 54    ② 20, 54, 23, 80, 35, 76  
     ③ 80, 23, 35, 76, 20, 54    ④ 20, 35, 23, 80, 54, 76

E: ①  $O(n^2)$ 和  $O(1)$                   ②  $O(n\log_2 n)$ 和  $O(1)$   
     ③  $O(n\log_2 n)$ 和  $O(n)$               ④  $O(n^2)$ 和  $O(n)$

正确答案: A: ③    B: ⑥    C: ⑥    D: ④    E: ②

分析:

在排序算法中,两两比较待排序的记录,当发现不满足顺序要求时,变更它们的相对位置,这就是冒泡排序。冒泡排序是一种交换排序,所以, A 是交换排序。每次在待排序的记录中选出关键字值最大(或最小)的记录,将它放到一列已排序记录的末尾,直到全部选完,这是选择排序,所以 B 是选择排序。则交换它们的位置,直到全部满足顺序要求为止。堆排序法是选择排序算法的改进算法。堆是一种数据结构。它将待排序的  $n$  个记录看成是具有  $n$  个结点的完全二叉树,树中结点满足:任何一个非叶结点的值(记录的关键字值)都大于等于(或小于、等于)它的子女结点的值。用该标准去衡量题中所给的 4 个序列:



很明显, 只有④满足堆的定义。

利用堆的性质可以进行排序: 先由给定的无序元素序列建立堆, 然后输出堆顶元素, 再用堆中最后一个元素代替顶元素, 将剩下的元素重新构成堆, 重复这一过程, 将得到按关键字值排序的输出序列。

堆排序的时间由两部分组成: 建立初始堆的时间和每输出一个记录后调整堆的时间, 前者为  $O(n\log_2 n)$ , 后者也为  $O(n\log_2 n)$ , 所以堆排序的平均执行时间为  $O(n\log_2 n)$ 。

堆排序算法需要的存储空间是存储  $n$  个结点的数组以及少量暂存单元, 因附加的存储单元很少, 仅需一个用于交换的附加存储结点, 所以为  $O(1)$ 。

【例 8-21】在二叉查找树中, 每个结点的关键字值 A, B 一棵二叉查找树, 即可得到排序序列。同一个结点集合, 可用不同的二叉查找树表示, 人们把平均查找长度最短的二叉查找树称作最佳二叉查找树, 最佳二叉查找树在结构上的特点是 C, D 不是二叉查找树, E 是最佳二叉查找树。

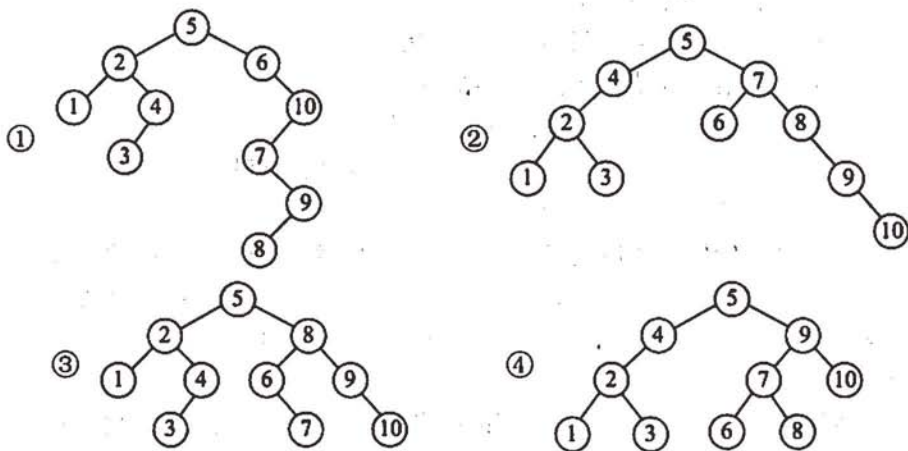
可选答案:

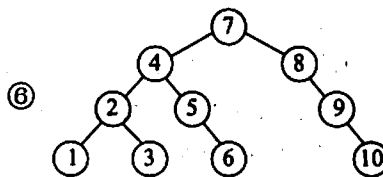
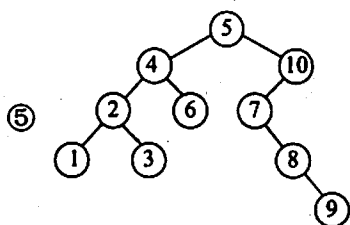
- A: ① 比左子树所有结点的关键字值大, 比右子树所有结点的关键字值小  
 ② 比左子树所有结点的关键字值大, 比右子树所有结点的关键字值大  
 ③ 比左右子树的所有结点的关键字值都大  
 ④ 与左子树所有结点的关键字值和右子树所有结点的关键字值无必然的大小关系

- B: ① 前序遍历 ② 中序遍历 ③ 后序遍历 ④ 层次序遍历

- C: ① 除最下二层可以不满外, 其余都是充满的  
 ② 除最下一层可以不满外, 其余都是充满的  
 ③ 每个结点的左右子树的高度之差的绝对值不大于 1  
 ④ 最下层的叶结点必须在最左边

D、E:





正确答案: A: ① B: ② C: ② D: ⑤ E: ③

分析:

二叉查找树是下列性质的二叉树: 二叉树中任何一个结点的关键字值都大于它的左子树中所有结点的关键字值, 而小于等于右子树中所有结点的关键字值。对二叉查找树进行中序遍历即按中序遍历左子树, 访问根, 按中序遍历右子树, 则可得到各结点关键字值的一个递增序列。根据二叉查找树的特点去检查题中所给的 6 个二叉树, 可以看到第⑤棵二叉树不满足条件, 因为左子树上有一结点值为 6, 大于根结点值 5。

同一个结点集合, 其插入二叉查找树的次序不同, 就构成不同的二叉查找树, 它们的平均查找长度也不相同, 把平均查找长度最短的二叉查找树称作最佳二叉查找树, 它在结构上的特点是: 除了最下一层可以不满足, 其他各层都是充满了的。上面已经说明, 题中所给的 6 棵树中, ①、②、③、④、⑥都是二叉查找树。而其中只有③是最佳二叉查找树。

【例 8-22】数据结构反映了数据元素之间的结构关系。链表是一种 A, 它对于数据元素的插入和删除 B。

通常查找线性表数据元素的方法有 C 和 E 两种方法, 其中 C 是一种只适合于顺序存储结构但 E 的方法; 而 D 是一种对顺序和链式存储结构均适用的方法。

可选答案:

- A: ① 顺序存储线性表      ② 非顺序存储非线性表  
      ③ 顺序存储非线性表      ④ 非顺序存储线性表
- B: ① 不需移动结点, 不需改变结点指针  
      ② 不需移动结点, 只需改变结点指针  
      ③ 只需移动结点, 不需改变结点指针  
      ④ 既需移动结点, 又需改变结点指针
- C: ① 顺序查找      ② 循环查找      ③ 条件查找      ④ 二分法查找
- D: ① 顺序查找      ② 随机查找      ③ 二分法查找      ④ 分块查找
- E: ① 效率较低的线性查找      ② 效率较低的非线性查找  
      ③ 效率较高的非线性查找      ④ 效率较高的线性查找

正确答案: A: ④ B: ② C: ④ D: ① E: ③

分析:

数据的逻辑结构反映了数据元素之间的逻辑关系, 与计算机无关; 数据的物理结构

也称存储结构,反映了数据在存储器中的存放方式。数据的存储结构主要有顺序存储结构和链式存储结构两种。最简单的数据结构是同类型数据元素的有限序列,称为线性表。采用链式存储结构的线性表称为链表。

链表中每个数据元素的存储单元称为结点,结点中除了数据项外,还包括指针(地址),指向其逻辑上相邻的元素。这样,逻辑上相邻的元素可以在不相邻的存储单元中,因此,链表是一种非顺序存储。

链表中删除或插入元素比较方便,无需改变结点的存储位置,只要修改几个结点的指针即可。而对顺序存储结构,插入或删除元素一般都要移动许多结点的位置,较费时间。

在线性表中查找指定元素常用顺序查找法和二分查找法。顺序查找法即是从第一个元素开始逐个元素地顺序查找,属于线性查找,比较的次数较多,效率较低;二分法查找适用于已排序的顺序存储线性表,也称折半查找,每次取中间一个元素进行判断,若已找到,则停止查找;若非所找,则需要决定取其前一半还是后一半继续查找,因此它不是线性查找,它比顺序查找的效率高一一些。

### 8.3 思考练习题及答案

1. 算法是对问题求解过程的一类精确描述,算法中描述的操作都是可以通过已经实现的基本操作在限定时间内执行有限次来实现的,这句话说明算法具有( )特性。

- A. 正确性      B. 确定性      C. 能行性      D. 健壮性

【解答】选C。能行性表明算法中所有的操作都是足够基本,能够在有限的时间内实现。

2. 一个算法的时间复杂度为  $(3n^2 + 2n\log_2 n + 4n - 7)/(5n)$ , 其时间复杂度为( )。

- A.  $O(n)$       B.  $O(n\log_2 n)$       C.  $O(n^2)$       D.  $O(\log_2 n)$

【解答】选A。根据n的最高阶决定该算法的渐进时间复杂度  $O(n^2/n)$ 。

3. 下述函数中渐进时间最小的是( )。

- A.  $T_1(n) = n\log_2 n + 100\log_2 n$       B.  $T_2(n) = n\log_2 n - 100\log_2 n$   
C.  $T_3(n) = n^2 - 100\log_2 n$       D.  $T_4(n) = 4n\log_2 n - 100\log_2 n$

【解答】选B或A。渐进时间复杂度表明当n趋于无穷大时,其  $T(n)$  的增长的趋势。首先排除C,因为它的最高阶显然大。再看其他三种情况。

$$\lim_{n \rightarrow \infty} (T_1/T_2) = 1, \quad \lim_{n \rightarrow \infty} (T_1/T_4) = 1/4$$

答案应选A或B。

4. 以下说法正确的是( )。

- A. 数据结构的逻辑结构独立于其存储结构。



- B. 数据结构的存储结构独立于该数据结构的逻辑结构。
- C. 数据结构的逻辑结构惟一地决定了该数据结构的存储结构。
- D. 数据结构仅由其逻辑结构和存储结构决定。

【解答】选 A。数据的逻辑结构面向应用，其存储结构面向计算机存储，根据存储的要求，同一逻辑结构可选用不同的存储结构，所以选 A。但数据结构的存储结构是数据逻辑结构的实现，必定与其逻辑结构有关，不可选 B。

5. 计算机数据处理的对象是具有不同结构的各种数据，可以访问的最小数据信息单位是 A，可以引用的最小命名数据单位是 B。

线性表是最简单的一种数据结构，有顺序和链接二种存储方式。线性表按链接方式存储时，每个结点的存储包括 C 两部分。

线性表的查找有 D 和 E 两种，但 E 只能用于顺序存储的情况。

可选答案：

- A: ① 数字                      ② 字符                      ③ 数据元素                      ④ 数据项  
B: ① 结点                      ② 记录                      ③ 数据元素                      ④ 数据项  
C: ① 数据值与符号              ② 数据与指针                  ③ 数据与表名                  ④ 头地址和尾地址  
D、E: ① 随机查找              ② 顺序查找                  ③ 二分法查找                  ④ 游览

【解答】选 A: ③    B: ④    C: ②    D: ②    E: ③

6. 判断下列叙述的对错。

- ① 所谓数据的逻辑结构是指数据元素之间的逻辑关系。
- ② 同一数据逻辑结构中的所有数据元素都具有相同的特性是指数据元素所包含的数据项的个数都相等。
- ③ 数据的逻辑结构与数据元素本身的内容和形式无关。
- ④ 数据结构是指相互之间存在一种或多种关系的数据元素的全体。
- ⑤ 从逻辑关系上讲，数据结构主要分为两大类：线性结构和非线性结构。
- ⑥ 线性表的逻辑顺序与物理顺序总是一致的。
- ⑦ 线性表的顺序存储表示优于链式存储表示。
- ⑧ 线性表若采用链式存储表示时所有存储单元的地址可连续可不连续。
- ⑨ 二维数组是其数组元素为线性表的线性表。
- ⑩ 每种数据结构都应具备三种基本运算：插入、删除和查找。

【解答】正确的有 (1)、(3)、(5)、(8)、(10)

7. 在一个长度为  $n$  的顺序表中向第  $i$  个元素 ( $0 \leq i \leq n$ ) 位置插入一个新元素时，需从后向前依次后移 (     ) 个元素。

- A.  $n-i$                       B.  $n-i+1$                       C.  $n-i-1$                       D.  $i$

【解答】选 A。

8. 在一个长度为  $n$  的顺序表中删除第  $i$  个元素 ( $0 \leq i \leq n-1$ ) 时，需要从前向后依次

前移 ( ) 个元素。

A.  $n-i$ B.  $n-i+1$ C.  $n-i-1$ D.  $i$ 

【解答】选 C。

9. 在一个长度为  $n$  的顺序表的表尾插入一个新元素的渐进时间复杂度为 ( )。

A.  $O(n)$ B.  $O(1)$ C.  $O(n^2)$ D.  $O(\log_2 n)$ 

【解答】选 B。因为无需移动元素。

10. 对长度为 10 的顺序表进行查找，若查找前面 5 个元素的概率相同，均为  $1/8$ ，查找后面 5 个元素的概率相同，均为  $3/40$ ，则查找到表中任一元素的平均查找长度为 ( )。

A. 5.5

B. 5

C.  $39/8$ D.  $19/4$ 

【解答】选 C。平均查找长度为  $\frac{1}{8}(1+2+3+4+5) + \frac{3}{40}(6+7+8+9+10) = \frac{15}{8} +$

$$\frac{3}{40}40 = \frac{39}{8}$$

11. 设有一个  $n \times n$  的对称矩阵 A，将其下三角部分按行存放在一个一维数组 B 中， $A[0][0]$  存放于 B[0] 中，那么第  $i$  行的对角元素  $A[i][i]$  存放于 B 中 ( ) 处。

A.  $(i+3)*i/2$ B.  $(i+1)*i/2$ C.  $(2n-i+1)*i/2$ D.  $(2n-i-1)*i/2$ 

【解答】选 A。第  $i$  行的对角元素  $A[i][i]$  存放于 B 中  $\sum_{k=0}^{i-1} (k+1) + i = \frac{i(i+1)}{2} + i =$

$$\frac{i(i+3)}{2} \text{ 处。}$$

12. 给定有  $n$  个元素的向量，逐个输入其中的元素值建立一个有序单链表的时间复杂度是 ( )。

A.  $O(1)$ B.  $O(n)$ C.  $O(n^2)$ D.  $O(n \log_2 n)$ 

【解答】选 C。向有序链表插入第  $i$  ( $0 \leq i < n$ ) 个元素值的时间复杂度为  $O(i)$ ，全部元素插入的时间复杂度为  $\sum_{i=0}^{n-1} O(i) = O(\sum_{i=0}^{n-1} i) = O(n^2)$ 。

13. 设二维数组  $F[i][j]$  的行下标为 1 至 5，列下标为 0 至 8，F 的每个数据元素均占 4 个字节。在按行存储的情况下，已知数据元素  $F[2][2]$  的第一个字节的地址是 1044，则  $F[3][4]$  和  $F[4][3]$  的第一个字节的地址分别为 A 和 B，而数组的第一个数据元素的第一个字节和数组最后一个元素的第一个字节的地址分别为 C 和 D。

对一般的二维数组 G 而言，当 E 时，其按行存储的  $G[i][j]$  的地址与按列存储的  $G[j][i]$  的地址相同。

可选答案：

A: ① 1088

② 1084

③ 1092

④ 1120

- B: ① 1092                      ② 1088                      ③ 1120                      ④ 1124  
 C: ① 1004                      ② 1044                      ③ 1000                      ④ 984  
 D: ① 1183                      ② 1176                      ③ 1164                      ④ 1187

- E: ① G 的列数与行数相同  
 ② G 的列的上界与 G 的行的上界相同  
 ③ G 的列的下界与 G 的行的下界相同  
 ④ G 的列的上下界与 G 的行的上下界相同

【解答】选 A: ① B: ③ C: ③ D: ② E: ④

根据已知条件, 行下标范围是 1~5, 列下标范围是 0~8, 则其行数为  $m = 5 - 1 + 1 = 5$ , 列数为  $n = 8 - 0 + 1 = 9$ , 每个元素占 4 个字节。在按行存储的情况下,

$$\text{Loc}(2, 2) = \text{Loc}(1, 0) + ((2 - 1) * 9 + 2 - 0) * 4 = 1044$$

可得数组第一个元素的首地址为  $\text{Loc}(1, 0) = 1000$ 。由此可得

$$\text{Loc}(3, 4) = \text{Loc}(1, 0) + ((3 - 1) * 9 + 4 - 0) * 4 = 1000 + 22 * 4 = 1088$$

$$\text{Loc}(4, 3) = \text{Loc}(1, 0) + ((4 - 1) * 9 + 3 - 0) * 4 = 1000 + 30 * 4 = 1120$$

最后一个元素的首地址为

$$\text{Loc}(5, 8) = \text{Loc}(1, 0) + ((5 - 1) * 9 + 8 - 0) * 4 = 1000 + 44 * 4 = 1176$$

设二维数组行的上下界为  $c_1 \sim d_1$ , 列的上下界为  $c_2 \sim d_2$ , 则  $m = d_1 - c_1 + 1$ ,  $n = d_2 - c_2 + 1$ , 按行存储时求  $G[I][J]$  地址的公式为  $\text{Loc}(I, J) = \text{Loc}(c_1, c_2) + ((I - c_1) * n + J - c_2) * \text{len}$ ; 按列存储时求  $G[J][I]$  地址的公式为  $\text{Loc}(J, I) = \text{Loc}(c_1, c_2) + ((J - c_2) * m + I - c_1) * \text{len}$ , 其中  $\text{len}$  是每个元素的字节数。若要求按行存储的  $G[I][J]$  的地址与按列存储的  $G[J][I]$  的地址相同, 必有

$$\text{Loc}(c_1, c_2) + ((I - c_1) * n + J - c_2) * \text{len} = \text{Loc}(c_1, c_2) + ((J - c_2) * m + I - c_1) * \text{len}$$

14. 二维数组 X 的行下标范围是 0~5, 列下标范围是 1~8, 每个数组元素占 6 个字节, 则该数组的体积为 A 个字节, 若已知 X 的最后一个元素的起始字节地址为 382, 则 X 的首地址 (即第一个元素的起始字节地址) 为 B, 记为  $X_d$ 。若按行存储, 则  $X[1][5]$  的起始地址是 C, 结束字节地址是 D。若按列存储, 则  $X[4][8]$  的起始字节地址为 E。

可选答案:

- A: ① 210                      ② 240                      ③ 288                      ④ 294  
 B: ① 0                      ② 6                      ③ 94                      ④ 100  
 C: ①  $X_d + 24$                       ②  $X_d + 72$                       ③  $X_d + 78$                       ④  $X_d + 144$   
 D: ①  $X_d + 29$                       ②  $X_d + 77$                       ③  $X_d + 83$                       ④  $X_d + 147$   
 E: ①  $X_d + 186$                       ②  $X_d + 234$                       ③  $X_d + 270$                       ④  $X_d + 276$

【解答】选 A: ③ B: ④ C: ② D: ② E: ④

根据已知条件, 共有  $m = 5 - 0 + 1 = 6$  行,  $n = 8 - 1 + 1 = 8$  列, 数组的体积为  $(6 \times 8) \times 6 = 288$  个字节。因为数组最后一个元素的起始字节地址为 382, 则 X 的首地址是  $382 - 288 + 6$

= 100。

若按行存储, 则  $X[1][5]$  的起始字节地址是  $X_d + (1 \times 8 + 5 - 1) \times 6 = X_d + 72$ , 结束字节地址是  $X_d + 72 + 6 - 1 = X_d + 77$ 。

若按列存储, 则  $X[4][8]$  的起始字节地址为  $X_d + ((8-1) \times 6 + 4) \times 6 = X_d + 276$ 。

15. 已知有一维数组  $A[0 \dots m \times n - 1]$ , 若要对应为  $m$  行,  $n$  列的矩阵, 则下面的对应关系 ( ) 可将元素  $A[k]$  ( $0 \leq k < m \times n$ ) 表示成矩阵的第  $i$  行, 第  $j$  列的元素。 ( $0 \leq i < m$ ,  $0 \leq j < n$ )。

A.  $i = k / n, j = k \% n$

B.  $i = k / m, j = k \% m$

C.  $i = k / n, j = k \% n$

D.  $i = k / m, j = k \% n$

【解答】选 C。若矩阵中元素在一维数组中按行存放, 矩阵元素的行/列号可以通过选项 C 求出。若矩阵元素在一维数组中按列存放, 则矩阵的行/列号可以通过 B 求出。

16. 已知  $A[n]$  为整数数组, 试写出实现下列运算的递归算法:

① 求数组  $A$  中的最大整数。

② 求  $n$  个整数的和。

③ 求  $n$  个整数的平均值。

【解答】

```
int MaxKey(int A[], int n){                //递归求最大值
    if(n==1) return A[0];
    int temp = MaxKey(A, n-1);
    return (A[n-1]>temp?A[n-1]:temp);
}
```

```
int Sum(int A[], int n){                    //递归求数组之和
    if(n==1) return A[0];
    else return A[n-1]+Sum(A, n-1);
}
```

```
float Average(int A[], int n){              //递归求数组的平均值
    if(n==1) return (float)A[0];
    else return ((float)A[n-1]+(n-1)*Average(A, n-1))/n;
}
```

17. 单选题

① 设链式栈中结点的结构为 (data, link), 且 top 是指向栈顶的指针。若想在链式栈的栈顶插入一个由指针 s 所指的结点, 则应执行下列哪一个操作?

A.  $top \rightarrow link = s;$

B.  $s \rightarrow link = top \rightarrow link; top \rightarrow link = s;$

C.  $s \rightarrow link = top; top = s;$

D.  $s \rightarrow link = top; top = top \rightarrow link;$

② 设链式栈中结点的结构为 (data, link), 且 top 是指向栈顶的指针。若想摘除链式栈的栈顶结点, 并将被摘除结点的值保存到 x 中, 则应执行下列哪一个操作?



A.  $x = \text{top} \rightarrow \text{data}; \text{top} = \text{top} \rightarrow \text{link};$

B.  $\text{top} = \text{top} \rightarrow \text{link}; x = \text{top} \rightarrow \text{data};$

C.  $x = \text{top}; \text{top} = \text{top} \rightarrow \text{link};$

D.  $x = \text{top} \rightarrow \text{data};$

③ 设循环队列的结构是

```
const int MaxSize = 100;
```

```
typedef int DataType;
```

```
typedef struct {
```

```
    DataType data[MaxSize];
```

```
    int front, rear;
```

```
} Queue;
```

若有一个 Queue 类型的队列 Q，试问判断队列满的条件应是下列哪一个语句？

A.  $Q.\text{front} == Q.\text{rear};$

B.  $Q.\text{front} - Q.\text{rear} == \text{MaxSize};$

C.  $Q.\text{front} + Q.\text{rear} == \text{MaxSize};$

D.  $Q.\text{front} == (Q.\text{rear} + 1) \% \text{MaxSize};$

④ 设循环队列的结构如(3)。若有一个 Queue 类型的队列 Q，试问应用下列哪一个语句计算队列元素个数？

A.  $(Q.\text{rear} - Q.\text{front} + \text{MaxSize}) \% \text{MaxSize};$

B.  $Q.\text{rear} - Q.\text{front} + 1;$

C.  $Q.\text{rear} - Q.\text{front} - 1;$

D.  $Q.\text{rear} - Q.\text{front};$

【解答】 选①C    ②A    ③D    ④A

18. 已知 f 为单链表的表首指针，链表中存储的都是整型数据，试写出实现下列运算的递归算法：

① 求链表中的最大整数。

② 求链表的结点个数。

③ 求所有整数的平均值。

【解答】

① 求链表中的最大整数

```
int Max(LinkedNode *f) {                //递归算法：求链表中的最大值
    if(f->link == NULL) return f->data;
    int temp = Max(f->link);
    return (f->data > temp ? f->data : temp);
}
```

## ② 求链表的结点个数

```
int Num(LinkedList *f){           //递归算法：求链表中结点个数
    if(f==NULL) return 0;
    return 1+Num(f->link);
}
```

## ③ 求所有整数的平均值

```
float Avg(LinkedList *f, int& n){ //递归算法：求链表中所有元素平均值
    if(f->link == NULL)
        {n=1; return(float)(f->data);}
    else{
        float Sum=Avg(f->link, n)*n; n++;
        return(f->data+Sum)/n;
    }
}
```

19. 表是一种数据结构，链表是一种 A。队列和栈都是线性表，栈的操作特性是 B，队列的操作特性是 C。今有一空栈 S，对下列待进栈的数据元素序列 a、b、c、d、e、f 依次进行进栈、进栈、出栈、进栈、进栈、出栈的操作，则此操作完成后，栈 S 的栈顶元素为 D，栈底元素为 E。

可选答案：

- |               |             |
|---------------|-------------|
| A: ① 非顺序存储线性表 | ② 非顺序存储非线性表 |
| ③ 顺序存储线性表     | ④ 顺序存储非线性表  |
| B: ① 随机进出     | ② 先进后出      |
| ③ 先进先出        | ④ 出优于进      |
| C: ① 随机进出     | ② 先进后出      |
| ③ 后进后出        | ④ 进优于出      |
| D: ① f        | ② c         |
| ③ a           | ④ b         |
| E: ① b        | ② c         |
| ③ a           | ④ d         |

【解答】选 A: ① B: ② C: ③ D: ② E: ③

## 20. 单选题

- ① 栈的插入和删除操作在 ( ) 进行。
- A. 栈顶      B. 栈底      C. 任意位置      D. 指定位置
- ② 当利用大小为 n 的数组顺序存储一个栈时，假定用  $top=n$  表示栈空，则向这个栈插入一个元素时，首先应执行 ( ) 语句修改 top 指针。
- A.  $top++$ ;      B.  $top--$ ;      C.  $top=0$ ;      D.  $top$ ;
- ③ 若让元素 1,2,3 依次进栈，则出栈次序不可能出现 ( ) 种情况。
- A. 3,2,1      B. 2,1,3      C. 3,1,2      D. 1,3,2
- ④ 当利用大小为 n 的数组顺序存储一个队列时，该队列的最大长度为 ( )。

A.  $n-2$ B.  $n-1$ C.  $n$ D.  $n+1$ 

⑤ 假定一个顺序存储的循环队列的队头和队尾指针分别为  $f$  和  $r$ , 则判断队空的条件为 ( )。

A.  $f+1 == r$ B.  $r+1 == f$ C.  $f == 0$ D.  $f == r$ 

⑥ 假定一个链式队列的队头和队尾指针分别为  $front$  和  $rear$ , 则判断队空的条件为 ( )。

A.  $front == rear$ B.  $front != NULL$ C.  $rear != NULL$ D.  $front == NULL$ 

【解答】 选 (1) A (2) B (3) C (4) B (5) D (6) D

21. 使用两个栈共享一片内存空间时, 当 ( ) 时, 才产生上溢。

A. 两个栈的栈顶同时到达这片内存空间的中心点

B. 其中一个栈的栈顶到达这片内存空间的中心点

C. 两个栈的栈顶在这片内存空间的某一位置相遇

D. 两个栈均不空, 且一个栈的栈顶到达另一个栈的栈底

【解答】 选 C。使用两个栈共享一片内存空间时, 通常将两个栈的栈底置于该内存空间两端, 在进栈时, 栈顶从两端向内, 直到它们在内存某个位置相遇才算栈满。

22. 若循环队列以数组  $Q[0 \dots m-1]$  作为其存储结构, 变量  $rear$  表示循环队列中队尾元素的实际位置, 其移动按  $rear = (rear+1) \bmod m$  进行, 变量  $length$  表示当前循环队列中的元素个数, 则循环队列的队首元素的实际位置是 ( )。

A.  $rear-length$ B.  $(rear-length+m) \bmod m$ C.  $(1+rear+m-length) \bmod m$ D.  $m-length$ 

【解答】 选 C。因为  $rear-length$  是实际队头的前一位置。 $rear-length+1$  可能是负值, 所以有  $(rear-length+1+m) \bmod m$ 。

23. 填空题

① 队列的插入操作在 ( ) 进行, 删除操作在 ( ) 进行。

② 栈又称为 ( ) 的表, 队列又称为 ( ) 的表。

③ 向一个顺序栈插入一个元素时, 首先使 ( ) 后移一个位置, 然后把待插入元素 ( ) 到这个位置上。

④ 从一个栈删除元素时, 需要前移一位 ( )。

⑤ 在一个循环队列  $Q$  中, 判断队空的条件为 ( ), 判断队满的条件为 ( )。

⑥ 在一个顺序栈中, 若栈顶指针等于 ( ), 则为空栈; 若栈顶指针等于 ( ), 则为满栈。

⑦ 在一个链式栈中, 若栈顶指针等于  $NULL$  则为 ( ); 在一个链式队列中, 若队头指针与队尾指针的值相同, 则表示该队列为 ( ) 或该队列 ( )。

⑧ 向一个链式栈插入一个新结点时, 首先把栈顶指针的值赋给 ( ), 然后把新结点的存储位置赋给 ( )。

⑨ 向一个循环队列中插入元素时,需要首先移动( ),然后再向所指位置( )新插入的元素。

⑩ 当用长度为  $n$  的数组顺序存储一个栈时,若用  $\text{top} == n$  表示栈空,则表示栈满的条件为( )。

⑪ 向一个栈顶指针为  $\text{top}$  的链式栈中插入一个新结点  $*p$  时,应执行( )和( )操作。

⑫ 从一个栈顶指针为  $\text{top}$  的非空链式栈中删除结点并不需要返回栈顶结点的值和回收结点时,应执行( )操作。

⑬ 假定  $\text{front}$  和  $\text{rear}$  分别为一个链式队列的队头和队尾指针,则该链式队列中只有一个结点的条件为( )。

⑭ 中缀表达式  $3*(x+2)-5$  所对应的后缀表达式为( )。

⑮ 后缀表达式 “ $4\ 5\ * \ 3\ 2\ + \ -$ ” 的值为( )。

#### 【解答】

① 队尾, 队头

② 后进先出, 先进先出

③ 栈顶指针, 写入

④ 栈顶指针

⑤  $\text{Q.front} == \text{Q.rear}, (\text{Q.rear}+1) \% \text{MaxSize}+1 == \text{Q.front}$

⑥  $-1, \text{MaxSize}-1$

⑦ 空栈, 空, 只含有一个结点

⑧ 新结点的指针域, 栈顶指针

⑨ 队尾指针, 写入

⑩  $\text{top} == 0$

⑪  $p \rightarrow \text{link} = \text{top}, \text{top} = p$

⑫  $\text{top} = \text{top} \rightarrow \text{link}$

⑬  $\text{front} == \text{rear} \ \&\& \ \text{front} != \text{NULL}$  或者  $\text{front} == \text{rear} \ \&\& \ \text{rear} != \text{NULL}$

⑭  $3 \times 2 + * 5 -$

⑮ 15

24. 设有一个递归算法如下

```
int fact(int n){
    if(n<=0) return 1;
    else return n*fact(n-1);
}
```

下面正确的叙述是( )。



- A. 计算  $\text{fact}(n)$  需要执行  $n$  次函数调用
- B. 计算  $\text{fact}(n)$  需要执行  $n+1$  次函数调用
- C. 计算  $\text{fact}(n)$  需要执行  $n+2$  次函数调用
- D. 计算  $\text{fact}(n)$  需要执行  $n-1$  次函数调用

【解答】选 B。主程序调用  $\text{fact}(n)$  称为外部调用，其他调用为内部调用，直到调用  $\text{fact}(0)$  为止， $\text{fact}(n)$  调用  $\text{fact}(n-1)$ ， $\text{fact}(n-1)$  调用  $\text{fact}(n-2)$ ， $\dots$ ， $\text{fact}(1)$  调用  $\text{fact}(0)$ ，内部调用  $n$  次，外部调用 1 次，总共  $n+1$  次。

25. 下面算法的时间复杂度为 ( )。

```
int f(unsigned int n){  
    if(n == 0 || n == 1) return 1;  
    else return n*f(n-1);  
}
```

- A.  $O(1)$                   B.  $O(n)$                   C.  $O(n^2)$                   D.  $O(n!)$

【解答】选 B。

26. 下面的程序段违反了算法的 ( ) 原则。

```
void sam()  
{ int n = 2;  
  while(!odd(n)) n += 2;  
  printf(n);  
}
```

- A. 有穷性                  B. 确定性                  C. 可行性                  D. 健壮性

【解答】选 A。在该算法中  $n$  永远都是偶数，导致 while 循环永远不能停止。

27. 一个递归的定义可以用递归的过程求解。通常递归的执行过程是 ( ) 的。

- A. 高效                  B. 低效                  C. 高质量                  D. 低质量

【解答】选 B。因为在递归过程中有许多重复计算。

28. 若广义表  $L = ((1, 2, 3))$ ，则  $L$  的长度和深度分别为 ( )。

- A. 1 和 1                  B. 1 和 2                  C. 1 和 3                  D. 2 和 2

【解答】选 B。表中只有一个表元素（一个子表），长度为 1；表中括号重数为 2，故深度为 2。

29. 在一棵二叉树的二叉链表中，空指针数等于非空指针数加 ( )。

- A. 2                  B. 1                  C. 0                  D. -1

【解答】选 A。假设二叉树中有  $n$  个结点，有  $2n$  个指针，其中非空指针只有  $n-1$  个，则空指针应有  $n+1$  个。

30. 在一棵度为 3 的树中，若有 2 个度为 3 的结点，有 1 个度为 2 的结点，则有 ( ) 个度为 0 的结点。

A. 4

B. 5

C. 6

D. 7

【解答】选 C。设二叉树中度为 0 的结点有  $n_0$  个，度为 1 的结点有  $n_1$  个，度为 2 的结点有  $n_2$  个，度为 3 的结点有  $n_3$  个，则二叉树的边数  $e = n - 1 = n_0 + n_1 + n_2 + n_3 - 1 = 3 * n_3 + 2 * n_2 + n_1$ ，有  $n_0 - 1 = 2 * n_3 + n_2$ 。按照题意， $n_0 = 2 * 2 + 1 + 1 = 6$ 。

31. 设结点  $x$  和  $y$  是二叉树中任意的两个结点。在该二叉树的先根遍历序列中  $x$  在  $y$  之前，而在其后根遍历序列中  $x$  在  $y$  之后，则  $x$  和  $y$  的关系是 ( )。

A.  $x$  是  $y$  的左兄弟B.  $x$  是  $y$  的右兄弟C.  $x$  是  $y$  的祖先D.  $x$  是  $y$  的后裔

【解答】选 C。设二叉树的先根遍历顺序为 DLR，后根遍历顺序为 LRD，根据题意，在先根遍历序列中  $x$  在  $y$  前，在后根遍历序列中  $x$  在  $y$  之后，若设  $x$  在根结点的位置， $y$  在其左子树或右子树中，即满足要求。

32. 在一棵完全二叉树中，其根的序号为 1，( ) 可判定序号为  $p$  和  $q$  的两个结点是否在同一层。

A.  $\lfloor \log_2 p \rfloor = \lfloor \log_2 q \rfloor$ B.  $\log_2 p = \log_2 q$ C.  $\lfloor \log_2 p \rfloor + 1 = \lfloor \log_2 q \rfloor$ D.  $\lfloor \log_2 p \rfloor = \lfloor \log_2 q \rfloor + 1$ 

【解答】选 A。若设结点序号为  $i$ ，则求结点所在层次的公式为  $\lfloor \log_2 i \rfloor$ ，设根所在层次为 0。

33. ( ) 从二叉树的任一结点出发到根的路径上，所经过的结点序列必按其关键字降序排列。

A. 二叉查找树

B. 最大堆

C. 最小堆

D. 平衡二叉树

【解答】选 C。

34. 若按层次顺序将一棵有  $n$  个结点的完全二叉树的所有结点从 1 到  $n$  编号，那么当 A 且  $i$  不等于 1 时，结点  $i$  的右兄弟是结点 B，否则结点  $i$  没有右兄弟。当 C 时，结点  $i$  的右子女是结点 D，否则结点  $i$  没有右子女。E 是完全二叉树结构的一个典型应用。

可选答案：

A、C: ①  $i \leq n/2$  ②  $i \leq (n+1)/2$  ③  $i \leq (n-1)/2$  ④  $i$  为奇数⑤  $i$  为偶数B、D: ①  $i+1$  ②  $i-1$  ③  $\lfloor i/2 \rfloor$  ④  $\lfloor (i+1)/2 \rfloor$ ⑤  $2i$  ⑥  $2i+1$ 

E: ① 博弈树 ② 决策树 ③ 堆排序 ④ 基数排序

⑤ 伙伴系统

【解答】选 A: ④ B: ① C: ③ D: ⑥ E: ③

如果将一棵有  $n$  个结点的完全二叉树自顶向下，同一层自左向右连续给结点编号 1, 2, ...,  $n-1$ ，然后按此结点编号将树中各结点顺序地存放于一个一维数组中，并简称编号

为  $i$  的结点为结点  $i$  ( $1 \leq i \leq n$ ), 则有以下关系:

- 若  $i = 1$ , 则结点  $i$  为根, 无双亲; 若  $i > 1$ , 则结点  $i$  的双亲为结点  $\lfloor i/2 \rfloor$ 。
- 若  $2i < n$ , 则结点  $i$  的左子女为结点  $2i$ 。
- 若  $2i+1 < n$ , 则结点  $i$  的右子女为结点  $2i+1$ 。
- 若结点编号  $i$  为偶数 (处于左子女位置) 且  $i < n$ , 则它的右兄弟为结点  $i+1$ 。
- 若结点编号  $i$  为奇数 (处于右子女位置) 且  $i \neq 1$ , 则它的左兄弟为结点  $i-1$ 。
- 结点  $i$  所在层次为  $\lfloor \log_2 i \rfloor$ 。

### 35. 填空题

- ① 对于一棵具有  $n$  个结点的树, 该树中所有结点的度数之和为 ( )。
- ② 假定一棵三叉树的结点个数为 50, 则它的最小深度为 ( ), 最大深度为 ( )。
- ③ 一棵深度为  $h$  的四叉树中, 最多含有 ( ) 结点。
- ④ 在一棵三叉树中, 度为 3 的结点数有 2 个, 度为 2 的结点数有 1 个, 度为 1 的结点数有 2 个, 那么度为 0 的结点数有 ( ) 个。
- ⑤ 一棵深度为 5 的满二叉树中的结点数为 ( ) 个, 一棵深度为 3 的满四叉树中的结点数为 ( ) 个。
- ⑥ 在一棵二叉树中, 假定度为 2 的结点有 5 个, 度为 1 的结点有 6 个, 则叶子结点数有 ( ) 个。
- ⑦ 对于一棵含有 40 个结点的理想平衡树, 它的深度为 ( )。
- ⑧ 若对一棵二叉树从 0 开始进行结点编号, 并按此编号把它顺序存储到一维数组  $a$  中, 即编号为 0 的结点存储到  $a[0]$  中, 其余类推, 则  $a[i]$  结点的左子女结点为 ( ), 右子女结点为 ( ), 双亲结点 ( $i \geq 1$ ) 为 ( )。
- ⑨ 对于一棵具有  $n$  个结点的二叉树, 对应二叉链表中指针总数为 ( ) 个, 其中 ( ) 个用于指向子女结点, ( ) 个指针空闲着。
- ⑩ 在一棵深度为  $h$  的丰满树 (理想平衡树) 中, 最少含有 ( ) 个结点, 最多含有 ( ) 个结点。
- ⑪ 在一个堆的顺序存储中, 若一个结点的下标为  $i$ , 则它的左子女结点的下标为 ( ), 右子女结点的下标为 ( )。
- ⑫ 在一个最小堆中, 堆顶结点的值是所有结点中的 ( ), 在一个最大堆中, 堆顶结点的值是所有结点中的 ( )。
- ⑬ 当向一个最小堆插入一个具有最小值的结点时, 该结点需要逐层 ( ) 调整, 直到被调整到 ( ) 位置为止。
- ⑭ 当从一个最小堆中删除一个结点时, 需要把 ( ) 结点填补到 ( ) 位置, 然后再按条件把它逐层 ( ) 调整。
- ⑮ 在 Huffman 编码中, 若编码长度只允许小于等于 4, 则除了已对两个字符编码为

0 和 10 外, 还可以最多对 ( ) 个字符编码。

【解答】

- |                    |   |                   |
|--------------------|---|-------------------|
| ① $n-1$            | ② 4, 49                                 | ③ $(4^{h+1}-1)/3$ |
| ④ 6                | ⑤ 63, 85                                | ⑥ 6               |
| ⑦ 5                | ⑧ $2*i+1, 2*i+2, \lceil (l-1)/2 \rceil$ | ⑨ $2n, n-1, n+1$  |
| ⑩ $2^h, 2^{h+1}-1$ | ⑪ $2i+1, 2i+2$                          | ⑫ 最小者, 最大者        |
| ⑬ 向上, 堆顶           | ⑭ 最后, 堆顶, 向下                            | ⑮ 4               |

36. 从下列有关树的叙述中, 选出五条正确叙述。

- ① 一棵二叉树的层次遍历方法只有前序法和后序法两种;
- ② 在 Huffman 树中, 外部结点(叶结点)的个数比内部结点个数多 1;
- ③ 完全二叉树一定是平衡二叉树;
- ④ 在二叉树的前序序列中, 若结点  $u$  在结点  $v$  之前, 则  $u$  一定是  $v$  的祖先;
- ⑤ 在二叉查找树中新结点总是作为叶结点插入;
- ⑥ 树的后序序列和其对应的二叉树的后序序列的结果是一样的;
- ⑦ 对 B 树删除某一关键字时, 可能会引起结点的分裂;
- ⑧ 在含有  $n$  个结点的树中, 边数只能是  $n-1$  条;
- ⑨ 最佳查找树就是查找效率最高的查找树;
- ⑩ 中序遍历二叉链表存储的二叉树时, 一般要用栈; 中序遍历穿线树时, 也必须使用栈。

【解答】 正确的有②、⑤、⑧、⑨

37. 二叉树 A。在完全二叉树中, 若一个结点没有 B, 则它必定是叶结点。每棵树都能惟一地转换成与它对应的二叉树。由树转换成的二叉树里, 一个结点  $N$  的左子女是  $N$  在原树里对应结点的 C, 而  $N$  的右子女是它在原树里对应结点的 D。二叉查找树的平均查找长度为 E。

可选答案:

- |                 |                 |
|-----------------|-----------------|
| A: ① 是特殊的树      | ② 不是树的特殊形式      |
| ③ 是两棵树的总称       | ④ 是只有二个根结点的树形结构 |
| B: ① 左子女结点      | ② 右子女结点         |
| ③ 子女            | ④ 兄弟            |
| C~D: ① 最左子女结点   | ② 最右子女结点        |
| ③ 最邻近的右兄弟结点     | ④ 最邻近的左兄弟结点     |
| ⑤ 最左的兄弟结点       | ⑥ 最右的兄弟结点       |
| E: ① $O(n)$     | ② $o(n)$        |
| ③ $O(\log_2 n)$ | ④ $o(\log_2 n)$ |

【解答】 选 A: ② B: ③ C: ① D: ③ E: ③

38. 判断下列叙述的对错。



- ① 树的后根遍历序列与对应二叉树的后序遍历序列是一样的。
- ② 若有一个结点是二叉树中某个子树的中序遍历结果序列的最后一个结点, 则它一定是该子树的前序遍历结果序列的最后一个结点。
- ③ 若有一个结点是二叉树中某个子树的前序遍历结果序列的最后一个结点, 则它一定是该子树的中序遍历结果序列的最后一个结点。
- ④ 若有一个叶子结点是二叉树中某个子树的中序遍历结果序列的最后一个结点, 则它一定是该子树的前序遍历结果序列的最后一个结点。
- ⑤ 若有一个叶子结点是二叉树中某个子树的前序遍历结果序列的最后一个结点, 则它一定是该子树的中序遍历结果序列的最后一个结点。

【解答】 正确的有④

39. 从具有  $n$  个结点的二叉查找树中查找一个元素时, 在最坏情况下进行成功查找的时间复杂度为 ( )。

- A.  $O(n)$                       B.  $O(1)$                       C.  $O(\log_2 n)$                       D.  $O(n^2)$

【解答】 选 A。最坏情况时二叉查找树成为单枝树。

40. 堆是一种数据结构, ( ) 是堆。

- A. (10, 50, 80, 30, 60, 20, 15, 18)  
B. (10, 18, 15, 20, 50, 80, 30, 60)  
C. (10, 15, 18, 50, 80, 30, 60, 20)  
D. (10, 30, 60, 20, 15, 18, 50, 80)

【解答】 选 B。画成完全二叉树的形式即可知 B 是最小堆。

41. 向具有  $n$  个结点的堆中插入一个新元素的时间复杂度为 ( )。

- A.  $O(1)$                       B.  $O(n)$                       C.  $O(\log_2 n)$                       D.  $O(n \log_2 n)$

【解答】 选 C。在堆中新元素插在最下层, 然后调用一个向上调整算法将其重新调整成堆。此算法的时间复杂度为  $O(\log_2 n)$ 。

42. 利用 3, 6, 8, 12 这四个值作为叶结点的权值生成一棵 Huffman 树, 该树的带权路径长度为 ( )。

- A. 55                      B. 29                      C. 58                      D. 38

【解答】 选 A。

43. 若一棵 Huffman 树共有 9 个结点, 则其叶子结点的个数为 (6)。

- A. 4                      B. 5                      C. 6                      D. 7

【解答】 选 B。Huffman 树中只有度为 0 和度为 2 的结点, 因此, Huffman 树中若有  $n$  个叶结点, 总结点有  $2n-1$  个, 如果  $2n-1=9$ , 则  $n=5$ 。

44. 每一棵树都能惟一地转换为它所对应的二叉树, 树的这种二叉树表示对树的运算带来很大的好处。遍历是树形结构的一种重要运算, 二叉树的基本组成部分是: 根(N)、左子树(L)和右子树(R)。因而二叉树的遍历次序有六种。最常用的是三种: 前序法

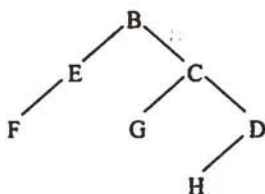
(即按 A 次序), 后序法 (即按 B 次序) 和中序法 (也称对称序法, 即按 C 次序)。这三种方法相互之间有关联。若已知一棵二叉树的前序序列是 BEFCGDH, 中序序列是 FEBGCHD, 则它的后序序列必是 D, 而且可得该二叉树所表示的树的先根次序序列是 E。

可选答案:

- A~C: ① RLN                      ② RNL                      ③ LRN                      ④ LNR  
           ⑤ NLR                      ⑥ NRL
- D、E: ① EFGHBCD                      ② FEGHDCB  
           ③ BCDEFGH                      ④ EFBGCHD  
           ⑤ BEFCGDH                      ⑥ FEGBHDC

【解答】选 A: ⑤    B: ③    C: ④    D: ②    E: ⑤

所以如果一棵二叉树的前序序列是 BEFCGDH, 中序遍历序列是 FEBGCHD, 那么可以推断出这棵二叉树的结构为:



其后序遍历序列必为 FEGHDCB, 其先根次序序列为 BEFCGDH。

本题的关键是正确掌握三种算法的递归定义。显然, 前序序列的第一个结点一定是树的根结点, 而后序序列的最后一个结点是根结点, 由题中可知根结点为 B, 答案中只 ② 符合 D 的要求, 那么 D 的答案一定是 ②。

45. 树是结点的集合, 它有 A 个根结点。二叉树有 B 个根结点, 按一定的规则, 任一棵树都可以转换成惟一对应的二叉树。二叉树的查找有深度优先和广度优先二类, 深度优先包括 C。当一棵二叉树的前序序列和中序序列分别是 HGEDBFCA 和 EGBDHFAC 时, 其后序序列必是 D, 层次序序列为 E。

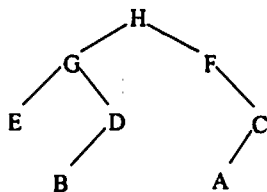
可选答案:

- A: ① 且只有 1                      ② 1 或多于 1                      ③ 0 或 1                      ④ 至少 2  
 B: ① 且只有 1                      ② 1 或多于 1                      ③ 0 或 1                      ④ 至少 2  
 C: ① 前序遍历、后序遍历、中序遍历  
       ② 前序遍历、后序遍历、层次序遍历  
       ③ 前序遍历、中序遍历、层次序遍历  
       ④ 中序遍历、后序遍历、层次序遍历  
 D: ① BDEAGFHC                      ② EBDGACFH                      ③ HGFEDCBA                      ④ HFGDEABC

E: ① BDEACGFH ② EBDGACFH ③ HGFEDCBA ④ HFGCDEAB

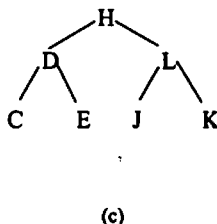
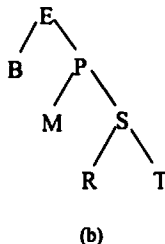
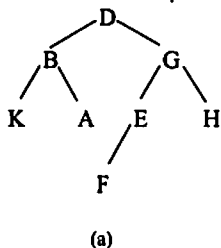
【解答】选 A: ① B: ③ C: ① D: ② E: ③

当一棵二叉树的前序序列和中序序列分别是 HGEDBFCA 和 EGBDHFAC 时, 对应的二叉树为:



其后序序列是 EBDGACFH, 层次序列为 HGFEDCBA。

46. 在下列二叉树中, 图(a)为 A 树, 图(b)为 B 树, 图(c)为 C 树。



可选答案:

A~C: ① 查找树

② 满二叉树

③ 平衡树但不是满二叉树

④ B 树

【解答】选 A: ③ B: ① C: ②

47. 从供选择的答案中选择与下面有关二叉树和森林的叙述中各括号相匹配的词句。

(1) 设二叉树有  $n$  个结点且根结点处于第 0 层, 则其高度为 ( A )。

(2) 设高度为  $h$  的空二叉树的高度为  $-1$ , 只有一个结点的二叉树的高度为 0, 若设二叉树只有度为 2 和度为 0 的结点, 则该二叉树中所含结点至少有 ( B ) 个。

(3) 设森林  $F$  中有 4 棵树, 第 1、2、3、4 棵树的结点个数分别为  $n_1$ 、 $n_2$ 、 $n_3$ 、 $n_4$ , 当把森林  $F$  转换成一棵二叉树后, 其根结点的右子树中有 ( C ) 个结点。

(4) 设森林  $F$  中有 4 棵树, 第 1、2、3、4 棵树的结点个数分别为  $n_1$ 、 $n_2$ 、 $n_3$ 、 $n_4$ , 当把森林  $F$  转换成一棵二叉树后, 其根结点的左子树中有 ( D ) 个结点。

(5) 将含有 82 个结点的完全二叉树从根结点开始顺序编号, 根结点为第 0 号, 其他结点自上向下, 同一层自左向右连续编号。则第 40 号结点的双亲结点的编号为 ( E )。

可选答案:

- A: ①  $n-1$                       ②  $\lceil \log_2(n+1) \rceil - 1$                       ③  $\lfloor \log_2 n \rfloor + 1$                       ④ 不确定  
 B: ①  $2h$                       ②  $2h-1$                       ③  $2h+1$                       ④  $h+1$   
 C~D: ①  $n_1-1$                       ②  $n_1+n_2+n_3$                       ③  $n_2+n_3+n_4$                       ④  $n_1$   
 E: ① 20                      ② 19                      ③ 81                      ④ 80

【解答】 选 A. ④    B. ③    C. ③    D. ①    E. ②

48. 每一棵树(森林)都惟一地对应到一棵二叉树,从而树(森林)的遍历次序与二叉树的遍历次序也有对应关系,二叉树的前序遍历对应到树的A,二叉树的后序遍历对应到树的B,二叉树的中序遍历对应到树的C。如果一棵二叉树结点的前序序列是 BEFCGDH,中序序列 EFBGCHD,则该二叉树结点的后序序列是D,该二叉树所对应的树(树林)的结点的层次序列是E。

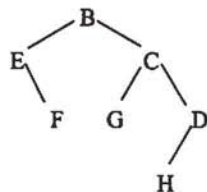
可选答案:

- A~C: ① 先根次序遍历                      ② 后根次序遍历  
           ③ 层次次序遍历                      ④ 与上述三种次序都不对应  
 D、E: ① EFGHBCD                      ② FEGHDCB                      ③ BCDEFGH  
           ④ EFBGCHD                      ⑤ BECFGDH

【解答】 选 A: ①    B: ④    C: ②    D: ②    E: ⑤

对树进行先根次序遍历的结果与对应二叉树的前序遍历结果一致;对树进行后根次序遍历的结果与对应二叉树的中序遍历结果一致。

由二叉树的前序序列和中序序列可惟一确定一棵二叉树。对于给定的前序序列 BEFCGDH 和中序序列 EFBGCHD,可得到二叉树如下,它的后序序列为 FEGHDCB,层次次序序列为 BECFGDH。



49. 二叉树的前序、中序和后序遍历法最适合采用A来实现。

二叉查找树中,由根结点到所有其他结点的路径长度的总和称为B,而使上述路径长度和达到最小的树称为C,它一定是D。

在关于树的几个叙述中,只有E是正确的。

可选答案:

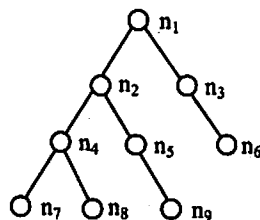
- A: ① 递归程序                      ② 迭代程序                      ③ 队列操作                      ④ 栈操作  
 B: ① 路径和                      ② 内部路径长度                      ③ 总深度                      ④ 深度和  
 C: ① B 树                      ② B+树                      ③ 丰满树                      ④ 穿线树  
 D: ① B 树                      ② 平衡树                      ③ 非平衡树                      ④ 穿线树  
 E: ① 用指针方式存储有  $n$  个结点的二叉树,至少要有  $n+1$  个指针  
       ②  $m$  阶 B 树中,每个非失败结点的子树棵数  $\geq \lfloor m/2 \rfloor$   
       ③  $m$  阶 B 树中,具有  $k$  棵子树的结点,必含有  $k-1$  个关键字



## ④ 平衡树一定是丰满树

【解答】选 A: ① B: ② C: ③ D: ② E: ③

50. 如图所示的二叉树, 有下列性质: 除叶结点外, 每个结点的值都大于其左子树上的一切结点的值, 并小于等于其右上树上一切结点的值。这是一棵 A 树。



现有一菲波那契数列  $\{a_n\}$ ,  $a_0 = a_1 = 1$ ,  $a_k = a_{k-1} + a_{k-2}$ ,  $k = 2, 3, \dots$ 。若把  $\{a_1, a_2, \dots, a_9\}$  填入该二叉树, 一般可以采用 B 遍历法遍历该二叉树上全部结点, 得到由结点的值组成的从小到大顺序排列的序列。对本题给出的二叉树图形填入  $\{a_1, \dots, a_9\}$  后, 其结点  $n_8$  的值为 C, 根结点的值为 D。若欲插入  $\{a_1, \dots, a_9\}$  的平均值, 则应该在 E 增加一个结点。

可选答案:

- A: ① 穿线树                      ② 最佳查找树                      ③ B-树                      ④ 二叉查找树  
 B: ① 前序                      ② 中序                      ③ 后序                      ④ 广度  
 C: ① 3                      ② 8                      ③ 21                      ④ 57  
 D: ① 8                      ② 21                      ③ 34                      ④ 66  
 E: ①  $n_2$  与  $n_4$  之间                      ②  $n_6$  下                      ③  $n_5$  与  $n_9$  之间                      ④  $n_9$  下

【解答】选 A: ④ B: ② C: ① D: ② E: ④

根据菲波那契数列和二叉查找树的定义, 应有如下的对应关系:

菲波那契数列	$a_1=1$	$a_2=2$	$a_3=3$	$a_4=5$	$a_5=8$	$a_6=13$	$a_7=21$	$a_8=34$	$a_9=55$
结点中序序列	$n_7$	$n_4$	$n_8$	$n_2$	$n_5$	$n_9$	$n_1$	$n_3$	$n_6$



因为这些数据的平均值为 15.78, 应插入在  $n_9$  与  $n_1$  之间。根据二叉查找树的插入规则, 应作为  $n_9$  的右子女结点插入。

51. 一棵深度为  $h$  (根结点所处层次为 0) 的满  $k$  叉树有如下性质: 第  $h$  层上的结点都是叶结点, 其余各层上每个结点都有  $k$  棵非空子树, 如果按层次自顶向下, 同一层自左向右, 顺序从 1 开始对全部结点进行编号, 则第  $j$  层 ( $j=0, 1, \dots, h$ ) 的结点个数是 A, 编号为  $i$  的结点的双亲结点 (若存在) 的编号是 B, 编号为  $i$  的结点的第  $m$  个子女结点 (若存在) 的编号是 C, 编号为  $i$  的结点有右兄弟的条件是 D, 其右兄弟结点的编号是 E。

可选答案:

- A~E: ①  $(k^h-1)/(k-1)$                       ②  $(k^{h+1}-1)/(k-1)$   
 ③  $\lfloor (i+k-2)/k \rfloor$                       ④  $\lceil \log_k(n(k-1)+1) \rceil - 1$   
 ⑤  $k^{i-1}$                       ⑥  $k^i$

⑦  $(i-1)k+m+1$ ⑧  $i-1$ ⑨  $i+1$ ⑩  $(i-1) \% k \neq 0$ 

【解答】 选 A: ⑥ B: ③ C: ⑦ D: ⑩ E: ⑨

52. 设一棵二叉树以二叉链表表示, 试编写有关二叉树的递归算法:

(1) 统计二叉树中度为 1 的结点个数。

(2) 统计二叉树中度为 2 的结点个数。

(3) 统计二叉树中度为 0 (叶结点) 的结点个数。

(4) 统计二叉树的深度。

(5) 统计二叉树的宽度, 即在二叉树的各层上, 具有结点数最多的那一层上结点总数。

(6) 从二叉树中删去所有叶结点。

(7) 计算二叉树中指定结点 \*p 所在层次。

(8) 计算二叉树中各结点中的最大元素的值。

(9) 以前序次序输出一棵二叉树所有结点的数据值及结点所在层次。

【解答】

(1) 统计二叉树中度为 1 的结点个数。

```
int Degrees1(BTNode *t){
    if(t == NULL) return 0;
    if(t->lchild != NULL && t->rchild == NULL ||
       t->lchild == NULL && t->rchild != NULL)
        return 1+Degrees1(t->lchild)+Degrees1(t->rchild);
    return Degrees1(t->lchild)+Degrees1(t->rchild);
}
```

(2) 统计二叉树中度为 2 的结点个数。

```
int Degrees2(BTNode *t){
    if(t == NULL) return 0;
    if(t->lchild != NULL && t->rchild != NULL)
        return 1+Degrees1(t->lchild)+Degrees1(t->rchild);
    return Degrees2(t->lchild)+Degrees2(t->rchild);
}
```

(3) 统计二叉树中度为 0 (叶结点) 的结点个数。

```
int leaves(BTNode *t){
    if(t == NULL) return 0;
    if(t->lchild == NULL && t->rchild == NULL) return 1;
    return leaves(t->lchild)+leaves(t->rchild);
}
```

## (4) 统计二叉树的深度。

```
int Depth(BTNode *t){
    if(t == NULL)return -1;
    int hl = Depth(t->lchild);
    int hr = Depth(t->rchild);
    return 1+(hl>hr?hl:hr);
}
```

(5) 统计二叉树的宽度，即在二叉树的各层上，具有结点数最多的那一层上结点总数。本题的想法是：先用前序遍历求出每一层的宽度，再求出最大宽度，即树的宽度。

## ① 求每一层宽度的算法：

```
int levelNumber(BTNode *t, int a[], int h){
    //求以*t 为根的子树中各层的宽度，存放在a[] 中，h是 *t 所在层次号，
    //要求在主程序中将 a[h]初始化为 0
    if(t != NULL){
        //若空树，不统计
        a[h] += 1; //第 h 层宽度加一
        levelNumber(t->lchild,a,h+1); //递归统计左子树中各层宽度
        levelNumber(t->rchild,a,h+1); //递归统计右子树中各层宽度
    }
}
```

## ② 求二叉树的宽度的算法。

```
int width(BTNode *t){
    int a[n+1], h = 0, i, wid;
    for(i=0; i<=n; i++)a[i] = 0; //统计数组 a[] 初始化
    levelNumber(t,a,h); //调用求各层宽度的算法
    wid = a[0];
    for(i=1; i<=n; i++)
        if(a[i] > wid) wid = a[i];
    return wid;
}
```

## (6) 从二叉树中删去所有叶结点。

```
void defoliate(BTNode *&t){
    if(t == NULL)return;
    if(t->lchild == NULL && t->rchild == NULL)
        {delete t; t = NULL;}
    else{
        defoliate(t->lchild);
        defoliate(t->rchild);
    }
}
```

```
defoliate(t->rchild);
```

(7) 计算二叉树中指定结点\*p所在层次。

```
int level(BTNode *t, BTNode *p){
    if(t == NULL) return -1;
    if(t == p) return 0;
    if((leftSubTreelevel = level(t->lchild, p)) > -1)
        return 1+leftSubTreelevel;
    if((rightSubTreelevel = level(t->rchild, p)) > -1)
        return 1+rightSubTreelevel;
    return -1;
}
```

(8) 计算二叉树中各结点中的最大元素的值。

```
Type MaxValue(BTNode *t, Type max){
    if(t != NULL){
        if(t->data > max) max = t->data;
        max = MaxValue(t->lchild, max);
        max = MaxValue(t->rchild, max);
    }
    return max;
}
```

(9) 以前序次序输出一棵二叉树所有结点的数据值及结点所在层次。

```
#include <stdio.h>
void nodePrint(BTNode *t, int i){
    if(t != NULL){
        printf("%d", t->data, "%d\n", i);
        nodePrint(t->lchild, i+1);
        nodePrint(t->rchild, i+1);
    }
}
```

53. 一个含有  $n$  个顶点和  $e$  条边的简单无向图, 在其邻接矩阵存储结构中共有 ( ) 个零元素。

A.  $e$

B.  $2e$

C.  $n^2 - e$

D.  $n^2 - 2e$

【解答】选 D。简单无向图的邻接矩阵是对称矩阵, 有  $e$  条边, 矩阵中将有  $2e$  个 1。



$N$  个顶点的邻接矩阵有  $n^2$  个矩阵元素, 其中零元素有  $n^2 - 2e$  个。

54. 若采用邻接矩阵存储简单有向图, 则其某一个顶点  $i$  的入度等于该矩阵 ( )。

- A. 第  $i$  行中值为 1 的元素个数
- B. 所有值为 1 的元素总数
- C. 第  $i$  行及第  $i$  列中值为 1 的元素总个数
- D. 第  $i$  列中值为 1 的元素个数。

【解答】选 D。邻接矩阵元素  $A[i][j]$  表示边  $\langle V_i, V_j \rangle$ , 矩阵列号表示边的终顶点号, 故顶点  $i$  的入度为矩阵第  $i$  列中值为 1 的元素的个数。

55. 判断下列有关图的叙述的正误。

① 用邻接矩阵存储一个图时, 在不考虑压缩存储的情况下, 所占用的存储空间大小只与图中的顶点个数有关, 而与图的边数无关。

② 邻接表只能用于有向图的存储, 邻接矩阵对于有向图和无向图的存储都适用。

③ 邻接矩阵只适用于稠密图(边数接近于顶点数的平方), 邻接表适用于稀疏图(边数远小于顶点数的平方)。

④ 有  $n$  ( $n \geq 1$ ) 个顶点的无向连通图最少有  $n-1$  条边。

⑤ 有  $n$  ( $n \geq 1$ ) 个顶点的有向强连通图最少有  $n$  条边。

⑥ 存储无向图的邻接矩阵是对称的, 因此只要存储邻接矩阵的下(上)三角部分就可以了。

⑦ 连通分量是无向图中的极小连通子图。

⑧ 强连通分量是有向图中的极大强连通子图。

⑨ 对任何用顶点表示活动的网络(AOV网)进行拓扑排序的结果都是惟一的。

⑩ 有回路的有向图不能完成拓扑排序。

⑪ 在 AOE 网络中一定只有一条关键路径。

⑫ 关键活动不按期完成就会影响整个工程的完成时间。

⑬ 任何一个关键活动提前完成, 那么整个工程将会提前完成。

⑭ 所有的关键活动都提前完成, 那么整个工程将会提前完成。

⑮ 任何一个关键活动延迟, 那么整个工程将会延迟。

【解答】正确的有 (1)、(3)、(4)、(5)、(6)、(8)、(10)、(12)、(14)、(15)

56. 给定数据结构  $(V, E)$ ,  $V$  为结点的有限集合,  $V = \{V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8\}$ ,  $E$  是  $V$  上关系的集合。

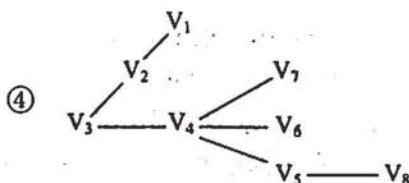
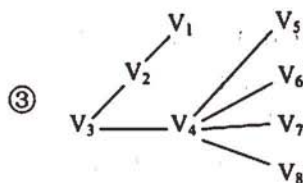
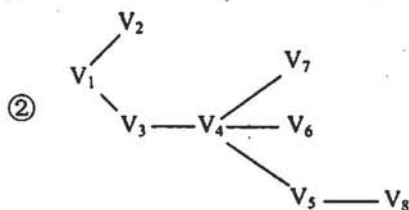
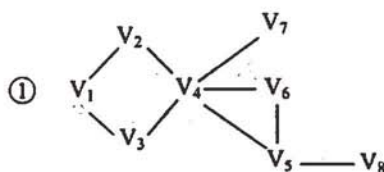
$E = \{(V_1, V_2), (V_3, V_4), (V_5, V_8), (V_5, V_6), (V_1, V_3), (V_4, V_7), (V_4, V_5), (V_2, V_4), (V_4, V_6)\}$

它所对应的图形是 A, 这是 B。

图的存储结构主要有邻接表和 C, 若用邻接表来存储一个图, 则需要保存一个 D 存储的顶点表和若干个 E 存储的关系表(又称边链表)。

可选答案:

A:



- B: ① 树                      ② 无向图                      ③ 有向图                      ④ 无回路图  
 C: ① 转移矩阵              ② 邻接矩阵                      ③ 状态矩阵                      ④ 优先矩阵  
 D: ① 顺序                      ② 链接                              ③ 散列                              ④ 分块  
 E: ① 顺序                      ② 链接                              ③ 散列                              ④ 索引

【解答】 选 A: ①    B: ②    C: ②    D: ①    E: ②

### 57. 填空题

- (1) 在一个无向图中, 所有顶点的度数之和等于所有边数的 ( ) 倍。
- (2) 在一个具有  $n$  个顶点的无向完全图中, 包含有 ( ) 条边, 在一个具有  $n$  个顶点的有向完全图中, 包含有 ( ) 条边。
- (3) 在一个具有  $n$  个顶点的无向图中, 要连通所有顶点则至少需要 ( ) 条边。
- (4) 表示图的三种存储结构为 ( )、( ) 和 ( )。
- (5) 对于一个具有  $n$  个顶点的图, 若采用邻接矩阵表示, 则矩阵大小为 ( )。
- (6) 对于一个具有  $n$  个顶点和  $e$  条边的有向图和无向图, 在其对应的邻接表中, 所含边结点分别为 ( ) 和 ( ) 条。
- (7) 在有向图的邻接表和逆邻接表表示中, 每个顶点的边链表中分别链接着该顶点的所有 ( ) 和 ( ) 结点。
- (8) 对于一个具有  $n$  个顶点和  $e$  条边的有向图和无向图, 若采用邻接多重表表示, 则存于顶点表中的边链表指针分别有 ( ) 和 ( ) 个, 所有边结点有 ( ) 个。
- (9) 对于一个具有  $n$  个顶点和  $e$  条边的无向图, 当分别采用邻接矩阵、邻接表和邻接多重表表示时, 求任一顶点度数的时间复杂度依次为 ( )、( ) 和 ( )。
- (10) 假定一个图具有  $n$  个顶点和  $e$  条边, 则采用邻接矩阵、邻接表和邻接多重表表示时, 其相应的空间复杂度分别为 ( )、( ) 和 ( )。

(11) 对用邻接矩阵表示的图进行任一种遍历时, 其时间复杂度为 ( ); 对用邻接表表示的图进行任一种遍历时, 其时间复杂度为 ( )。

(12) 对于一个具有  $n$  个顶点和  $e$  条边的连通图, 其生成树中的顶点数和边数分别为 ( ) 和 ( )。

- 【解答】 (1) 2 (2)  $n(n-1)/2$ ,  $n(n-1)$   
 (3)  $n-1$  (4) 邻接矩阵, 邻接表, 邻接多重表  
 (5)  $n^2$  (6)  $e$ ,  $2e$   
 (7) 出边, 入边 (8)  $2n$ ,  $n$ ,  $e$   
 (9)  $O(n)$ ,  $O(e/n)$ ,  $O(e)$  (10)  $O(n^2)$ ,  $O(n+e)$ ,  $O(n+e)$   
 (11)  $O(n^2)$ ,  $O(e)$  (12)  $n$ ,  $n-1$

58. 应用 Prim 算法求解连通网络的最小生成树问题。

(1) 针对下图所示的连通网络, 试按如下格式给出在构造最小生成树过程中顺序选出的各条边。

(始顶点号, 终顶点号, 权值)

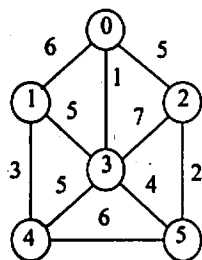
( , , )

( , , )

( , , )

( , , )

( , , )



(2) 下面是 Prim 算法的实现, 中间有 5 个地方缺失, 请阅读程序后将它们补上。

```
const int MaxInt = INT_MAX;           //INT_MAX 的值在<limits.h>中
const int n = 6;                      //图的顶点数, 应由用户定义
typedef int AdjMatrix[n][n];          //用二维数组作为邻接矩阵表示
typedef struct {                      //生成树的边结点
    int fromVex, toVex;               //边的起点与终点
    int weight;                       //边上的权值
} TreeEdgeNode;
typedef TreeEdgeNode MST[n-1];        //最小生成树定义

void PrimMST(AdjMatrix G, MST T, int rt){
    //从顶点 rt 出发构造图 G 的最小生成树 T, rt 成为树的根结点
    TreeEdgeNode e; int i, k = 0, min, minpos, v;
    for (i = 0; i < n; i++)           //初始化最小生成树 T
        if (i != rt){
            T[k].fromVex = rt;
            ①;
        }
    }
```

```

    T[k++].weight = G[rt][i];
}
for (k = 0; k < n-1; k++){           //依次求 MST 的候选边
    ②;
    for (i = k; i < n-1; i++)         //遍历当前候选边集合
        if (T[i].weight < min)       //选具有最小权值的候选边
            { min = T[i].weight; ③; }
    if (min == MaxInt)                //图不连通, 出错处理
        { cerr << "Graph is disconnected!" << endl; ④; }
    e = T[minpos]; T[minpos] = T[k]; T[k] = e;
    v = T[k].toVex;
    for (i = k+1; i < n-1; i++)       //修改候选边集合
        if (G[v][T[i].toVex] < T[i].weight){
            T[i].weight = G[v][T[i].toVex];
            ⑤;
        }
    }
}

```

## 【解答】

(1) 选出的边顺序为: 【括号中的内容为: (始顶点号, 终顶点号, 权值)】

(0, 3, 1), (3, 5, 4), (5, 2, 2), (3, 4, 5) 或 (3, 1, 5), (4, 1, 3), (1, 4, 3)

(2) 应填入的语句是: ① T[k].toVex = i;      ② min = MaxInt;  
 ③ minpos = i;      ④ exit(1);      ⑤ T[i].fromVex = v;

## 59. 填空题

(1) 每次从无序表中取出一个元素, 把它插入到有序表中的适当位置, 此种排序方法叫做 ( ) 排序; 每次从无序表中挑选出一个最小或最大元素, 把它交换到有序表的一端, 此种排序方法叫做 ( ) 排序。

(2) 每次直接或通过基准元素间接比较两个元素, 若出现逆序排列时就交换它们的位置, 此种排序方法叫做 ( ) 排序; 每次使两个相邻的有序表合并成一个有序表的排序方法叫做 ( ) 排序。

(3) 在直接选择排序中, 记录比较次数的时间复杂度为 ( ), 记录移动次数的时间复杂度为 ( )。

(4) 在堆排序的过程中, 对  $n$  个记录建立初始堆需要进行 ( ) 次调整运算, 由初始堆到堆排序结束, 需要对树根结点进行 ( ) 次调整运算。

(5) 在堆排序的过程中, 对任一分支结点进行调整运算的时间复杂度为 ( ), 整个堆排序过程的时间复杂度为 ( )。

(6) 假定一组记录的关键字为 (46, 79, 56, 38, 40, 84), 则利用堆排序方法建立的初始堆



为 ( )。

(7) 快速排序在平均情况下的时间复杂度为 ( ), 在最坏情况下的时间复杂度为 ( )。

(8) 快速排序在平均情况下的空间复杂度为 ( ), 在最坏情况下的空间复杂度为 ( )。

(9) 在快速排序方法中, 进行每次划分时, 是从当前待排序区间的 ( ) 向 ( ) 依次查找出处于逆序的元素并交换之, 最后将基准元素交换到一个确定位置, 从而以该位置把当前区间划分为前后两个子区间。

(10) 假定一组记录的关键字为 (46,79,56,38,40,80), 对其进行快速排序的一次划分的结果为 ( )。

(11) 在二路归并排序中, 对  $n$  个记录进行归并的趟数为 ( )。

(12) 在归并排序中, 进行每趟归并的时间复杂度为 ( ), 整个排序过程的时间复杂度为 ( ), 空间复杂度为 ( )。

(13) 对 20 个记录进行归并排序时, 共需要进行 ( ) 趟归并, 在第三趟归并时是把长度为 ( ) 的有序表两两归并为长度为 ( ) 的有序表。

(14) 假定一组记录的关键字为 (46,79,56,38,40,80), 对其进行归并排序的过程中, 第二趟归并后的结果为 ( )。

### 【解答】

(1) 插入, 选择

(2) 交换, 二路归并

(3)  $O(n^2)$ ,  $O(n)$

(4)  $\lfloor n/2 \rfloor$ ,  $n-1$

(6)  $O(\log_2 n)$ ,  $O(n \log_2 n)$

(6) (84,79,56,38,40,46)

(7)  $O(n \log_2 n)$ ,  $O(n^2)$

(8)  $O(\log_2 n)$ ,  $O(n)$

(9) 两端, 中间

(10) [38 40]46[56 79 84]

(11) 4, 4

(12)  $O(n)$ ,  $O(n \log_2 n)$ ,  $O(n)$

(13) 5, 4, 8

(14) [38 46 56 79][40 84]

60. 当所有待排序记录的关键字都相等, 计算下列排序方法的运行时间:

(1) 直接插入排序;

(2) 堆排序;

(3) 起泡排序;

(4) 直接选择排序。

### 【解答】

(1)  $n-1$

(2)  $2 \lfloor n/2 \rfloor$  ( $n$  为奇数),  $2 \lfloor n/2 \rfloor - 1$  ( $n$  为偶数)

(3)  $n-1$

(4)  $n(n-1)/2$

61. 填空题

(1) 对  $n$  个不同的关键字进行起泡排序, 在 ( ① ) 情况下关键字比较次数最小; 在 ( ② ) 情况下关键字比较次数最大。

(2) 快速排序在 ( ③ ) 情况下最不利于发挥其长处, 在 ( ④ ) 情况下最

易发挥其长处。

(3) 将5个不同的数据进行排序,最少需要( ⑤ )次数据比较,最多需要( ⑥ )次数据比较。

(4) 堆的形状是一棵( ⑦ )树。

(5) 就平均时间而言,( ⑧ )排序最好。

【解答】

- ① 排序对象已经按其关键字从小到大排好
- ② 排序对象已经按其关键字从大到小排好,现要求全部逆转
- ③ 排序对象已经按其关键字从小到大排好
- ④ 每次划分对象定位后,左侧子序列与右侧子序列长度相同的
- ⑤ 5
- ⑥ 25
- ⑦ 完全二叉
- ⑧ 快速

62. 在内排序的过程中,通常需要对待排序的关键字集合进行多遍扫描。采用不同的排序方法,会产生不同的排序中间结果,设要将集合 { tang, deng, an, wan, shi, bai, fang, li } 中的关键字按升序重新排列,则 A 是起泡排序一趟扫描的结果。B 是初始步长为4的 shell 排序一趟扫描的结果。C 是二路归并排序一趟扫描的结果。D 是以第一个元素为分界元素的快速排序一趟扫描的结果。E 是堆排序初始建堆的结果。

可选答案:

- A~E: ① deng, tang, an, wan, bai, shi, fang, li  
② an, deng, bai, li, shi, tang, fang, wan  
③ deng, an, tang, shi, bai, fang, li, wan  
④ deng, tang, an, wan, shi, bai, fang, li  
⑤ an, bai, deng, fang, li, shi, tang, wan  
⑥ an, tang, deng, wan, shi, bai, fang, li  
⑦ li, deng, an, fang, shi, bai, tang, wan  
⑧ shi, bai, an, li, tang, deng, fang, wan

【解答】选 A: ③ B: ⑧ C: ① D: ⑦ E: ②

63. 按排序策略分类,冒泡排序属于 A。对 n 个记录的文件进行排序时,如果待排序文件中的记录初始时已完全排好序,则冒泡排序过程中需进行 B 次元素值的比较, C 次元素值的交换。如果待排序文件的记录初始时为所要求次序的逆序,则冒泡排序过程中需进行 D 次元素值的比较, E 次元素的交换。

可选答案:

- A: ① 插入排序 ② 选择排序 ③ 交换排序 ④ 分配排序 ⑤ 归并排序

- B~E: ① 0            ②  $n-1$             ③  $n$             ④  $n+1$             ⑤  $n(n-1)/2$   
           ⑥  $n(n+1)/2$     ⑦  $n(n-1)$     ⑧  $n(n+1)$

【解答】选 A: ③    B: ②    C: ①    D: ⑤    E: ⑤

64. 堆是一种特殊的数据结构, A 是一个堆, 堆排序是一种 B 排序,  $m$  个元素进行堆排序时, 其时间复杂性为 C。

排序的算法很多, 若按排序的稳定性和不稳定性分类, 则 D 是不稳定排序。

外排序是指 E。

可选答案:

- A: ① 19, 75, 34, 26, 97, 56    ② 97, 26, 34, 75, 19, 56  
       ③ 19, 56, 26, 97, 34, 75    ④ 19, 34, 26, 97, 56, 75  
 B: ① 归并            ② 交换            ③ 选择            ④ 插入  
 C: ①  $O(m)$             ②  $O(m^2)$             ③  $O(\log_2 m)$             ④  $O(m \log_2 m)$   
 D: ① 冒泡排序    ② 归并排序    ③ 直接插入排序    ④ 希尔(shell)排序  
 E: ① 用机器指令直接对硬盘中需排序数据排序  
       ② 把需排序数据, 用其他大容量机器排序  
       ③ 把外存中需排序数据一次性调入内存, 排好序后, 再输回外存  
       ④ 对外存中大于内存允许空间的需要排序的数据, 通过多次内外存间的交换实现排序

【解答】选 A: ④    B: ③    C: ④    D: ④    E: ④

65. 给定结点的关键字序列 (F、B、J、G、E、A、I、D、C、H), 对它按字母的字典顺序进行排列, 采用不同方法, 其最终结果相同。但中间结果是不同的。

Shell 排序的第一趟扫描 (步长为 5) 结果应为 A。

冒泡排序 (大数下沉) 的第一趟起泡的效果是 B。

快速排序的第一趟结果是 C。

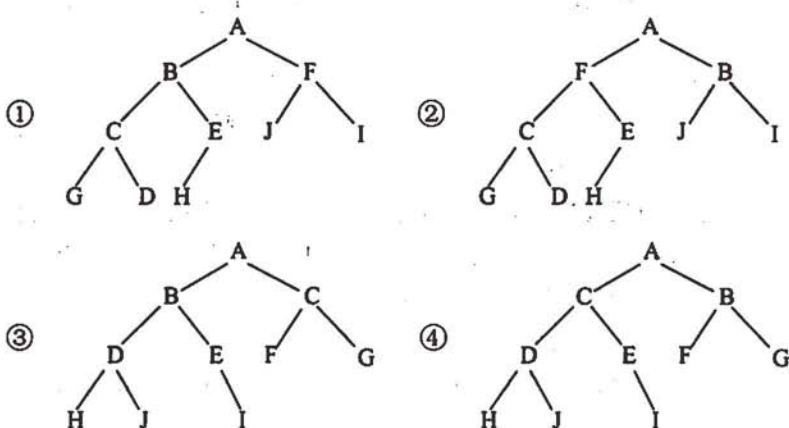
二路归并排序的第一趟结局是 D。

若以层次序序列来建立对应的完全二叉树, 采用渗透法建堆, 其第一趟建的堆是 E。

可选答案:

- A: ① (B、F、G、J、A、D、I、E、H、C)  
       ② (B、F、G、J、A、E、D、I、C、H)  
       ③ (A、B、D、C、E、F、I、J、G、H)  
       ④ (C、B、D、A、E、F、I、G、J、H)  
 B: ① (A、B、D、C、F、E、I、J、H、G)  
       ② (A、B、D、C、E、F、I、H、G、J)

- ③ (B、F、G、E、A、I、D、C、H、J)  
 ④ (B、F、G、J、A、E、D、I、C、H)  
 C: ① (C、B、D、A、F、E、I、J、G、H)  
 ② (C、B、D、A、E、F、I、G、J、H)  
 ③ (B、A、D、E、F、G、I、J、H、C)  
 ④ (B、C、D、A、E、F、I、J、G、H)  
 D: ① (B、F、G、J、A、E、D、I、C、H)  
 ② (B、A、D、E、F、G、I、J、H、C)  
 ③ (A、B、D、C、E、F、I、J、G、H)  
 ④ (C、B、D、C、F、E、J、I、H、G)  
 E:



【解答】 选 A: ③ B: ③ C: ② D: ① E: ②

66. 希尔排序、简单选择排序、快速排序和堆排序是不稳定的排序方法，试举例说明。

【解答】

- (1) 希尔排序 { 512 275 275\* 061 } 增量为 2  
 { 275\* 061 512 275 } 增量为 1  
 { 061 275\* 275 512 }  
 (2) 直接选择排序 { 275 275\* 512 061 } i = 1  
 { 061 275\* 512 275 } i = 2  
 { 061 275\* 512 275 } i = 3  
 { 061 275\* 275 512 }  
 (3) 快速排序 { 512 275 275\* }  
 { 275\* 275 512 }



- (4) 堆排序
- |        |      |      |       |                         |
|--------|------|------|-------|-------------------------|
| { 275  | 275* | 061  | 170 } | 已经是最大堆, 交换 275 与 170    |
| { 170  | 275* | 061  | 275 } | 对前 3 个调整                |
| { 275* | 170  | 061  | 275 } | 前 3 个最大堆, 交换 275* 与 061 |
| { 061  | 170  | 275* | 275 } | 对前 2 个调整                |
| { 170  | 061  | 275* | 275 } | 前 2 个最大堆, 交换 170 与 061  |
| { 061  | 170  | 275* | 275 } |                         |

67. 若对 27 个元素只进行三趟多路归并排序, 则选取的归并路数为 ( )。

- A. 2                      B. 3                      C. 4                      D. 5

【解答】选 B。若设参加归并的元素有  $m$  个, 归并路数为  $k$ , 则归并趟数等于  $\lceil \log_k m \rceil$ 。根据题意, 有  $\lceil \log_k 27 \rceil = 3$ , 得到  $k = 3$ 。

68. 以关键字比较为基础的排序算法在最坏情况下的计算时间下界为  $O(n \log n)$ 。下面的排序算法中, 最坏情况下计算时间可以达到  $O(n \log n)$  的是 A; 该算法采用的设计方法是 B。

- A: ① 归并排序              ② 插入排序              ③ 选择排序              ④ 冒泡排序  
B: ① 分治法              ② 贪心法              ③ 动态规划方法              ④ 回溯

【解答】选 A: ①      B: ①

69. 设顺序存储的某线性表共有 123 个元素, 按分块查找的要求等分为 3 块。若对索引表采用顺序查找方法来确定子块, 且在确定的子块中也采用顺序查找方法, 则在等概率的情况下, 分块查找成功的平均查找长度为 ( )。

- A. 21                      B. 23                      C. 41                      D. 62

【解答】选 B。在分块查找 (索引非顺序查找) 中  $ASL = ASL_{\text{index}} + ASL_{\text{subblock}}$ , 因此有

$$ASL = (3+1)/2 + (123/3+1)/2 = 2+21 = 23$$

70. 判断下列各叙述的正误。

① 在索引顺序结构上实施分块查找, 在等概率情况下, 其平均查找长度不仅与子表个数有关, 而且与每一个子表中的记录个数有关。

② 在索引顺序结构的查找中, 对索引表既可以采取顺序查找, 也可以采用折半查找。

③ 理想情况下, 在 Hash 表中查找一个元素的时间复杂度为  $O(1)$ 。

④ 在 Hash 法中, 一个可用的 Hash 函数必须保证绝对不产生冲突。

⑤ 在 Hash 法中采取拉链法来解决冲突时, 其装载因子的取值一定在  $(0,1)$  之间。

⑥ 在 Hash 法中采取开地址法来解决冲突时, 一般不要立刻做物理删除, 否则在查找时会发生错误。

【解答】正确的有①、②、③、⑥

71. 填空题

- ① 在索引表中, 每个索引项至少包含有 ( ) 域和 ( ) 域这两项。
- ② 假定一个线性表为 {12, 23, 74, 55, 63, 40, 82, 36}, 若按  $\text{key} \% 3$  条件进行划分, 使得同一余数的元素成为一个子表, 则得到的三个子表分别为 ( )、( ) 和 ( )。
- ③ 假定一个线性表为 ("abcd", "baabd", "bcef", "cfg", "ahij", "bkwte", "ccdt", "aayb"), 若按照字符串的第一个字母进行划分, 使得同一个字母被划分在一个子表中, 则得到的 a, b, c 三个子表的长度分别为 ( )、( ) 和 ( )。
- ④ 在索引表中, 若一个索引项对应数据记录表中的一个表项, 则称此索引为 ( ) 索引, 若对应数据记录表中的若干表项, 则称此索引为 ( ) 索引。
- ⑤ 假定对长度  $n = 100$  的线性表进行索引顺序查找, 并假定每个子表的长度均为  $\sqrt{n}$ , 则进行索引顺序查找的平均查找长度为 ( ), 时间复杂度为 ( )。
- ⑥ 若对长度  $n = 10000$  的线性表进行二级索引存储, 每级索引表中的索引项是下一级 20 个表项的索引, 则一级索引表的长度为 ( ), 二级索引表的长度为 ( )。
- ⑦ 假定要对长度  $n = 100$  的线性表进行 Hash 存储, 并采用拉链法处理冲突, 则对于长度  $m = 20$  的 Hash 表, 每个 Hash 地址的同义词子表(单链表)的长度平均为 ( )。
- ⑧ 在线性表的 Hash 存储中, 装载因子  $\alpha$  又称为装载系数, 若用  $m$  表示 Hash 表的长度,  $n$  表示待 Hash 存储的元素的个数, 则  $\alpha$  等于 ( )。

## 【解答】

- ① 关键字值, 子表地址域      ② (12, 63, 36), (55, 40, 82), (23, 74)
- ③ 3, 3, 2      ④ 稠密, 稀疏
- ⑤  $11, O(\sqrt{n})$       ⑥ 500, 25
- ⑦ 5      ⑧  $n/m$

## 72. 单选题

- ① 在 10 阶 B 树中根结点所包含的关键字个数最多为 ( ), 最少为 ( )。
- A. 1      B. 2      C. 9      D. 10
- ② 当对一个线性表  $R[60]$  进行索引顺序查找(分块查找)时, 若共分成了 10 个子表, 每个子表有 6 个表项。假定对索引表和数据子表都采用顺序查找, 则查找每一个表项的平均查找长度为 ( )。
- A. 7      B. 8      C. 9      D. 10
- ③ 既希望较快地查找又便于线性表动态变化的查找方法是 ( )。
- A. 顺序查找      B. 折半查找      C. Hash 查找      D. 索引顺序查找
- ④ Hash 函数有共同的性质, 则函数值应当以 ( ) 概率取其值域的每一个值。
- A. 最大      B. 最小
- C. 平均      D. 同等
- ⑤ 设 Hash 地址空间为  $0 \sim m-1$ ,  $k$  为表项的关键字, Hash 函数采用除余法, 即  $\text{Hash}(k) = k \% p$ 。为了减少发生冲突的频率, 一般取  $p$  为 ( )。







## 第9章 常用算法设计方法

### 9.1 内容提要

算法是为了解决某个问题而设计的步骤和方法。有了算法，就可以据此编写程序，在计算机上调试运行，最后得到问题的解。显然，即使某人对某种计算机语言（比如 C 语言）非常熟悉，如果自己不会根据要解决的问题设计算法，也是不可能编写出程序解决相应的问题。而设计算法往往是许多人惧怕的事情，他们一般在给出问题的数学表达（实际上就是算法）或文字叙述的解题步骤后，会很快编写出程序；而对于只是简单叙述的问题，比如，请编写程序给出从  $n$  个数中取  $k$  个的全部解，会让许多人觉得无从下手。而一个优秀的程序员或高级程序员必须具备这种素质，在这方面进行充分的训练，对于给出的问题，不但要能设计出算法，而且要能设计出简单易行的优秀算法。当然这不是一朝一夕就能实现的，需要艰苦的学习和磨炼。

解决同一个问题不同的人（甚至同一个人）可能会写出几种不同的算法。但算法有优劣之分，往往有这样的情况：同样一个问题，根据一种算法编写的程序可能需要几天甚至几个星期才能得到最终的解；而根据另一个好的算法编写的程序可能只需要几小时甚至几分钟就能得到同样的解。追求的最优算法当然应该是计算次数最少、所需存储空间最小的，但这两者往往不可兼得。在解决实际问题时，经常出现需要“以时间换空间”或“以空间换时间”的情况，这跟所用的计算机的内存和速度有关，需要折中考虑。

经常采用的算法设计技术主要有：迭代法、穷举搜索法、递推法、递归法、贪婪法、回溯法、分治法、动态规划法等。

#### 9.1.1 迭代法

##### 内容要点

(1) 迭代法是用来解决数值计算问题中的非线性方程（组）求解或最优解的一种算法设计方法，是许多方法的总称，包括了简单迭代法、对分法、梯度法、牛顿法等。

(2) 其主要思想是：从某个点出发，通过某种方式求出下一个点，此点应该离要求解的点（方程的解）更近一步，当两者之差接近到可以接收的精度范围时，就认为找到了问题的解。但问题的关键是要保证其收敛性。

##### 学习难点

程序员和高级程序员教程上给出的将方程（组） $f(x)=0$  变换成  $x=g(x)$  再迭代求解的方法简单易懂，在此不再重复。但此方法的缺点是：

(1) 每次只能求出方程的一个解, 而且需要人工先给出方程解的近似初值, 若初值选不好, 经常会得不到解。因此程序中应该加入一个迭代次数计数器, 当到一定的次数后, 就认为找不到解, 不再循环迭代下去, 防止“死循环”;

(2) 在变换成  $x=g(x)$  格式时, 不易保证其收敛性, 因为有时收敛性并不易证明。

因此这种并不是解方程的最好方法。下面介绍对分法和梯度法, 前者可用来求出方程  $f(x)=0$  在区间  $[a,b]$  内的全部单实根, 后者可用来求解非线性方程组的实根。

### 1. 对分法

对分法用来求出方程  $f(x)=0$  在区间  $[a,b]$  内的全部单实根, 是从端点  $x_0=a$  开始, 以  $h$  为步长, 逐步向后搜索。方法是:

对于每一个子区间  $[x_i, x_{i+1}]$  (其中  $x_{i+1}=x_i+h$ ):

- (1) 若  $f(x_i)=0$ , 则  $x_i$  为一个实根, 从  $x_i+h/2$  开始再向后搜索;
- (2) 若  $f(x_{i+1})=0$ , 则  $x_{i+1}$  为一个实根, 从  $x_{i+1}-h/2$  开始再向后搜索;
- (3) 若  $f(x_i) \cdot f(x_{i+1}) > 0$ , 则说明当前子区间内无实根, 从  $x_{i+1}$  开始再向后搜索;
- (4) 若  $f(x_i) \cdot f(x_{i+1}) < 0$ , 则说明当前子区间内有实根。此时, 反复将子区间减半, 直到找到一个实根或子区间长度小于  $\epsilon$  (给定的精度) 为止。

(5) 从  $x_{i+1}$  开始再向后搜索; 重复上述过程, 直到遇到区间右端点  $b$  为止。

这种方法的优点是: 只要在区间  $[a,b]$  内有根, 则一定能找到。

### 2. 梯度法

梯度法又称为最速下降法, 通常用来求非线性实函数方程组  $f_i(x_0, x_1, \dots, x_{n-1})=0$  ( $i=0, 1, \dots, n-1$ ) 的一组实根。方法是:

设方程组为  $f_i(x_0, x_1, \dots, x_{n-1})=0$  ( $i=0, 1, \dots, n-1$ ), 目标函数为:

$$F = F(x_0, x_1, \dots, x_{n-1}) = \sum_{i=0}^{n-1} f_i^2$$

- (1) 选取一组初值  $x_0, x_1, \dots, x_{n-1}$ ;
- (2) 计算目标函数  $F$ ;
- (3) 若  $F < \epsilon$ , 则  $X=(x_0, x_1, \dots, x_{n-1})^T$  即为方程组的一组实根, 过程结束; 否则继续下一步;

(4) 计算目标函数  $F$  在  $(x_0, x_1, \dots, x_{n-1})$  点的偏导数

$$\frac{\partial F}{\partial x_i} = 2 \sum_{j=0}^{n-1} f_j \cdot \frac{\partial f_j}{\partial x_i} \quad i = 0, 1, \dots, n-1$$

$$\text{再计算: } D = \sum_{j=0}^{n-1} \left( \frac{\partial F}{\partial x_j} \right)^2$$

- (5) 计算:  $x_i - \lambda \frac{\partial F}{\partial x_i} \Rightarrow x_i \quad i = 0, 1, \dots, n-1$  其中  $\lambda = F/D$

### (6) 转步骤(2)。

若上述过程中, 遇到  $D = 0$ , 则说明遇到了目标函数的局部极值点, 可改变初值再试。梯度法看上去比较复杂, 但它是目前用来解决工程问题使用非常普遍的一种方法。

## 9.1.2 穷举搜索法

### 内容要点

(1) 穷举搜索法是按某种顺序对所有的可能逐个进行验证, 从中找出符合要求条件的作为问题的解。

(2) 此方法通常需要用多重循环(循环重数取决于变量个数)来实现, 把每个变量的每个值都测试是否满足所给定的条件, 是则找到了问题的一个解。

(3) 这种方法简单易行, 尤其是对于一时想不出更好的算法来解决的问题, 不失为一种补救的好方法。

(4) 此方法只能解决变量个数非常有限, 而且每个变量可取值个数也很有限的情况。否则会造成“指数爆炸”, 计算机上难以实现。

### 学习难点

(1) 要找出问题所要满足的条件, 并用逻辑表达式表达清楚。

(2) 许多问题表面上可以用穷举搜索法实现, 但由于计算量等方面的原因, 你不得不将方法优化, 否则计算机上不可能实现。而方法的优化是一件不容易做的事。

## 9.1.3 递推法

### 内容要点

(1) 递推法是利用所解问题本身所具有的一种性质: 递推关系来求得问题解的一种行之有效的方法。

(2) 具体做法是: 对于一个问题, 可以根据  $N = n$  之前的一步  $n-1$  或多步 ( $n-1$ 、 $n-2$ 、 $n-3$ , ...) 的结果推导出  $n$  时的解:  $f(n) = F(f(n-1), f(n-2), \dots)$ , 而  $N = 0$  ( $1, 2, \dots$ ) 的初值  $f(0), f(1), \dots$  往往是直接给出或可以直观得出的。

(3) 但  $N = n$  的解与它前面的几步的结果相关必须在一开始就是确定的。比如说两步, 这样在计算的过程中只需记住这前两步的结果  $R_1, R_2$ , 下一步的结果  $R$  可以由这两步的结果推导得到:  $R = F(R_1, R_2)$ , 这时让  $R_1 = R_2, R_2 = R$ , 向前传递, 则为求再下一步的结果做好了准备。这也正是递推法名字的由来。而若问题与前三步相关, 则在计算的过程中需要记住这前三步的结果  $R_1, R_2, R_3$ , 下一步的结果  $R$  由这前三步的结果计算得到:  $R = F(R_1, R_2, R_3)$ , 这时让  $R_1 = R_2, R_2 = R_3, R_3 = R$ , 向前传递。

(4) 递推法是一种简单有效的方法, 一般用此方法编写出程序的执行效率很高, 可以用来解决有前效相关性、但前效相关顺序确定而且个数不多且是定数的问题。

(5) 递推法与递归法的关系是: 任何可以用递推法解决的问题, 可以很方便地用

递归法写出程序解决。反之，许多用递归法解决的问题，不能用递推法解决。这是因为递归法利用递归时的压栈，可以有任意长度和顺序的前效相关性，这是递推法所不具备的。

#### 学习难点

(1) 递推法的关键问题是找出递推关系式，并确定初值。对于某些问题这不是一件容易做到的事，可以结合归纳法来实现。

(2) 编程时要注意递推向前传递变量值时的时序，比如  $R_1=R_2$ ,  $R_2=R_3$ ,  $R_3=R$ ，不能出现颠倒，否则不成递推关系。

### 9.1.4 递归法

#### 内容要点

(1) 递归法是描述算法的一种强有力的方法，其思想是：将  $N=n$  时不能得出解的问题，设法递归（压栈）转化为求  $n-1, n-2, \dots$  的问题，一直到  $N=0$  或  $1$  的初始情况，由于初始情况的解可以给出或方便得到，因此开始层层退栈得到  $N=2, 3, \dots, n$  时的解，得到最终结果。

(2) 用递归法写出的程序简单易读，但与递推法编写的程序比，往往效率不高，因为每一次的递归函数调用都要压栈退栈。同时递归次数也不能无限制，因为每一次的递归函数调用都要压栈占用内存，而计算机的内存是有限的。

#### 学习难点

(1) 写递归程序的关键是，对于一个问题，要找出其递归关系和初始值。可以利用归纳法把一个问题归纳总结出递归式，加上初始条件，就可以编写递归函数。

对于单变量的问题  $f(n)$ ，步骤是：

(a) 当  $n=1$  或  $0$  时，可以得到  $f(n)$  的值；

(b) 假设小于等于  $n-1$  时，都可以得到  $f(n)$  的值；

(c) 则对于  $n$ ，找出  $f(n)$  与  $f(n-1)$ ,  $f(n-2)$ ,  $\dots$  的关系： $f(n) = F(f(n-1), f(n-2), \dots)$ ；

(d) 开始编写递归程序。

通常的程序格式是：

```
int f(int n)
{
    if (n==0 或 1) return 某个值或动作;
    else return F(f(n-1), f(n-2), ...);
}
```

递归方法利用递归时的压栈，可以有任意长度的前效相关性（记忆），因此有时甚至可以用来代替循环语句构成循环。比如，若编写一个函数  $f(n)$ ，要求在函数中不使用循环变量打印出  $1, 2, 3, \dots, n$  这个数列。就可以用递归法实现：



```
int f(int n)
{ if (n > 1) f(n-1);
  printf("%d, ", n);
}
```

如果将打印语句放在递归语句之前, 输出结果将是:  $n, n-1, \dots, 2, 1$ 。因此在编写递归程序时一定要注意这个递归与操作的时序问题。用递归方法实现二叉树的遍历时, 就是利用递归与操作的不同时序, 形成了前序遍历、中序遍历和后序遍历。

另一个要注意的是, 上面的程序中的 `printf` 语句前没有 `else`, 就是说, 这个 `printf` 语句对于任意  $n$  都要执行, 不同于一般的递归程序中的 `if` 语句二择一(不能省略 `else` 语句)。

(2) 对于两个变量的递归问题  $f(m, n)$ , 步骤是:

(a) 当  $m$  (或  $n$ ) 等于 1 或 0 时, 或  $m=n$  时, 可以得到  $f$  的初值;

(b) 假设  $\leq m, n-1$  (或  $\leq m-1, n$ ) 时, 可以得到  $f$  的值;

(c) 则对于  $(m, n)$ , 找出  $f(m, n)$  与前面的项  $f(m, n-1), f(m, n-2), \dots, f(m-1, n), f(m-1, n), \dots, f(m-1, n-1), f(m-1, n-2), \dots$  的关系:  $f(m, n) = F(f(m, n-1), f(m, n-2), \dots)$ ;

(d) 开始编写递归程序。

通常的程序格式是:

```
int f(int m, int n)
{ if (n(或 m) == 0 或 1 || m=n) return 某个值或动作;
  else return F(f(m, n-1), f(m, n-2), ...);
}
```

### 9.1.5 回溯法

#### 内容要点

(1) 回溯法又称为试探法, 是找到问题解的一种搜索策略: 就是在用某种方法找出解的过程中, 若中间项结果满足所解问题的条件, 则一直沿这个方向搜索下去, 直到无路可走或无结果, 则开始回溯, 改变其前一项的方向(或值)继续搜索。若其上一项的方向(或值)都已经测试过, 还无路可走或无结果, 则再继续回溯到更前一项, 改变其方向(或值)继续搜索。若找到了一个符合条件的解, 则停止或输出这个结果继续搜索; 否则继续回溯下去, 直到回溯到问题的开始处(不能再回溯), 仍没有找到符合条件的解, 则表示此问题无解或已经找到了全部的解。

(2) 用回溯法可以求得问题的一个解或全部解。为了不重复搜索已经找到过的解, 需要采取某种方法来标记。最常用的方法是用栈(数组)来记录已经找到的解或搜索过的情况(路径), 也可以用位置指针、值的排列顺序等来标记。

#### 学习难点

(1) 用回溯法求问题的一个解, 找到的解不一定是最优的。在编写这类程序时, 要

注意记录(或标记)好中间的每一个项的值,以便回溯,回溯到起始处表示无解,可以给出提示信息。

(2) 用回溯法求某个问题的全部解时,要注意在找到一组解时,及时输出或记录下来并统计解的个数,马上改变当前项值继续找下一组解,防止找到的解重复。回溯到起始处表示找到了问题的全部解(尽管可能一组解也没输出),提示找到了  $n$  ( $n$  可能为 0) 组解。

### 9.1.6 贪婪法

#### 内容要点

(1) 贪婪法是一种不追求最优解,只希望最快得到较为满意解的方法。“贪婪”是指在局部最优情况下,就决定当前项的值。因为贪婪法不去搜索所有的可能情况,所以可以快速找到符合要求的解。

(2) 贪婪法仅从你给出的起始值(尽管初值不尽合理)开始搜索,而不再改变起始值去搜索各种可能的整体情况,所以贪婪法不要回溯。

(3) 由于贪婪法不回溯,所以当你给出的起始值不合理时,此方法可能找不到问题的解,尽管问题的解实际存在。

(4) 贪婪法只能求出问题的某个解,而不可能给出所有的解。

#### 学习难点

尽管贪婪法不追求最优解而追求最快找到解,并且由于“贪婪”,有时最终可能会找不出解,但在对某个问题写算法编写程序时,也要尽量考虑解的好坏,以及尽可能找到解,否则程序就失去了意义。

### 9.1.7 分治法

#### 内容要点

(1) 分治法是被广泛使用的一种算法设计方法。基本思想是:首先把难以求解的大问题分解成许多容易求解的小问题,在多个小问题解的基础上综合得到大问题的解。

(2) 由于分治法需要把大问题分解成许多小问题,而小问题若仍不够小得不到解时,需要再分解成更小的问题,因此分治法经常需要与递归法结合使用。

#### 学习难点

(1) 有些问题不容易找出把大问题分解成小问题的“分治”方法,而且分治方法不合理时,有可能找不到问题的解。

(2) 由于分治法有时需要与递归法结合使用,所以最终的解决问题的算法可能就是按递归的方法来设计,只是要贯串分治的思想。

(3) 分治时的边界要清晰,防止重叠。

## 9.1.8 动态规划法

### 内容要点

某些复杂问题不能简单分解成几个小问题、在小问题解的基础上简单综合得到问题的解,因为这样费时费力,重复度高。因此需要引入一个数组,把所有子问题解存在其中,问题的最后解将从这个序列中得到,往往是选取概率最大的、得分最高的子问题的解综合得到问题的最后解,这就是动态规划法的基本思想。

### 学习难点

动态规划法往往需要给出一个求最佳值的递推函数。

## 9.2 例题分析

### 9.2.1 迭代法

#### 1. 对分法

对分法的通用程序是:

```
#include <stdio.h>
#include <math.h>

void bisection(double a, double b, double h, double eps, double x[], int m[2])
/* a 为区间的左端点, b 为区间的右端点, h 为步长, eps 为精度要求,
x 为所求的全部实根, m[0] 是实根个数的估计数, m[1] 是找到的实根的实际个数 */
{
    extern double f(); /* 要解的方程式 f(x) */
    int n, js; double z, y, z1, y1, z0, y0;
    n=0; z=a; y = f(z);
    while (z <= b+h/2.0 && n != m[0])
    {
        if (fabs(y) < eps) /* 判断是否 f(xi)=0
                                注意: 因为误差的关系, 不能用: if (y == 0.0) */
        {
            n++; x[n-1]=z; z += h/2.0; y=f(z); }
        else
        {
            z1=z+h; y1=f(z1);
            if (fabs(y1) < eps) /* 判断是否 f(xi+1)=0 */
            {
                n++; x[n-1]=z1; z=z1+h/2.0; y=f(z); }
            else if (y*y1 > 0.0) /* 判断是否 f(xi) · f(xi+1)>0 */
            { y=y1; z=z1; }
            else
            {
                js=0;
                while (js == 0)
                {
                    if (fabs(z1-z) < eps)
```

```

    { n++; x[n-1]=(z1+z)/2.0; z=z1+h/2.0; y=f(z); js=1; }
    else
    { z0=(z1+z)/2.0; y0=f(z0);
      if (fabs(y0) < eps)
      { x[n]=z0; n++; js=1; z=z0+h/2.0; y=f(z); }
      else if (y*y0 < 0.0) /* 判断是否  $f(x_i) \cdot f(x_{i+1}) < 0$  */
      { z1=z0; y1=y0; }
      else { z=z0; y=y0; }
    }
  }
}

m[1]=n; return;
}

```

**【例 9-1】** 求方程  $f(x)=x^6-5x^5+3x^4+x^3-7x^2+7x-20=0$  在区间 $[-2,5]$ 内的全部单实根。取基本步长  $h=0.2$ ,  $\text{eps}=10^{-6}$ , 程序为:

```

double f(double x)
{ return (((((x-5.0)*x+3.0)*x+1.0)*x-7.0)*x+7.0)*x-20.0; }

main()
{ int i; static int m[2]={6,0}; static double x[6]; double eps=1e-6;
  bisection(-2.0,5.0,0.2,eps,x,m);
  printf("M=%d\n",m[1]);
  for (i=0; i<m[1]; i++) printf("x(%d)=%12.6e\n",i,x[i]);
}

```

运行结果是:

M=2

$x(0)=-1.402463e+000$

$x(1)=4.333755e+000$

## 2. 梯度法

对分法的通用程序是:

```

#include <stdio.h>
int grads (int n,double eps,double x[],double y[],int js)
/* n 为方程个数; eps 为控制精度要求;
   x 函数调用时存放一组初值  $x_0, x_1, \dots, x_{n-1}$ , 返回时存放一组解;
   y 为临时数组; js 为允许最大迭代次数; 函数返回值为实际迭代次数 */
{ extern double f();

```



```

int l,j; double f,d,s;
l = js; f = f(x,y,n);
while (f >= eps)
{ l--; if (l==0) return(js);
  d=0.0;
  for (j=0; j<n; j++) d += y[j]*y[j];
  if (d+1.0 == 1.0) return(-1);
  s=f/d;
  for (j=0; j<n; j++) x[j] -= s*y[j];
  f = f(x,y,n);
}
return(js-l);
}

```

【例 9-2】 求下面非线性方程组的一组实根：

$$\begin{cases} f_0 = x_0 - 5x_1^2 + 7x_2^2 + 12 = 0 \\ f_1 = 3x_0x_1 - x_0x_2 - 11x_0 = 0 \\ f_2 = 2x_1x_2 + 40x_0 = 0 \end{cases}$$

取初值为(1.5, 6.5, -5.0)，精度要求为 1e，最大迭代次数为 500，程序为：

```

main()
{ int i,js=500; double eps=1e-6, y[3]; static double x[3]={1.5,6.5,-5.0};
  i= grads (3,eps,x,y,js);
  if (i>0 && i<js) for (i=0; i<3; i++) printf("x(%d)=%12.6e\n",i,x[i]);
}

double f(double x, double y, int n)
{ double z,f1,f2,f3,df1,df2,df3;
  f1=x[0] - 5.0*x[1]*x[1] + 7.0*x[2]*x[2] + 12.0;
  f2=3.0*x[0]*x[1] + x[0]*x[2] - 11.0*x[0];
  f3=2.0*x[1]*x[2] + 40.0*x[0];
  z=f1*f1 + f2*f2 + f3*f3; df1=1.0; df2=3.0*x[1] + x[2] - 11.0; df3=40.0;
  y[0]=2.0 * (f1*df1 + f2*df2 + f3*df3); df1=10.0*x[1]; df2=3.0*x[0];
  df3=2.0*x[2];
  y[1]=2.0 * (f1*df1 + f2*df2 + f3*df3); df1=14.0*x[2]; df2=x[0];
  df3=2.0*x[1];
  y[2]=2.0 * (f1*df1 + f2*df2 + f3*df3);
  return(z);
}

```

运行结果是：

x(0)=1.000188e+000

$x(1)=5.000432e+000$

$x(2)=-4.000388e+000$

### 9.2.2 穷举搜索法

【例 9-3】 求解“百钱买百鸡”，公鸡每只 5 钱，母鸡每只 3 钱，小鸡 3 只 1 钱。设公鸡数为  $x$ ，母鸡数为  $y$ ，小鸡数为  $z$ ，则可以得到下面的整数不定方程组：

$$\begin{cases} x+y+z=100 \\ 5*x+3*y+z/3=100 \end{cases}$$

利用穷举搜索法可以写出下面的程序：

```
# include <stdio.h>
void main()
{   int x,y, z;
    for (x=0; x<=100; x++)
        for (y=0; y<=100; y++)
            for (z=0; z<=100; z++)
                if (x+y+z==100 && 5*x + 3*y + z/3 == 100)
                    printf("%d %d %d\n", x,y,z);
}
```

运行结果是：

0 25 75

3 20 77

4 18 78

7 13 80

8 11 81

11 6 83

12 4 84

虽然上面的程序很简单，但分析一下我们会知道，此程序为三重循环，循环次数为： $101^3=1030301$ ，为  $10^6$  量级。如果改为“万钱买万鸡”，循环次数将达到  $10^{12}$  量级，即使目前最快的 PC 机也需要很长时间。这正是穷举搜索法的缺点所在，如果不对算法进行优化，计算量将以指数速度增长。而优化有时并不是一件非常困难的事情。比如，就针对“百钱买百鸡”问题，可以设想一下，即使所有的钱去买公鸡，最多也就 20 只，而所有的钱去买母鸡，最多也就 33 只，当  $x$ 、 $y$  的值确定后， $z$  的值就自然确定了，而不再是一个变量，不需要这层循环。考虑了这些因素，写出了与上面程序等效的程序：

```
# include <stdio.h>
void main()
```

```

{   int x,y, z;
    for (x=0; x<=20; x++)
        for (y=0; y<=33; y++)
            {   z = 100 - x - y;
                if (5*x + 3*y + z/3 == 100) printf("%d %d %d\n", x,y,z);
            }
}

```

这个程序的循环次数只有：21\*34=714，比上面的程序的计算量降低了3个数量级。而“万钱买万鸡”问题照此修改程序，它的计算量降低到：2000\*3333=6.666×10<sup>6</sup>，比原来降低了5个数量级！可见即使是穷举搜索法，只要注意优化，还是大有潜力可挖的。

再来分析一下上面程序的运行结果，一共有7组解，但第2、4、6组解的小鸡数并不是3的倍数，所以这些解并不正确。这是由于z/3整除引起的。需要在程序中再加入一个条件判断z是否是3的倍数。修改后的程序如下：

```

# include <stdio.h>
void main()
{   int x,y, z;
    for (x=0; x<=20; x++)
        for (y=0; y<=33; y++)
            {   z = 100 - x - y;
                if (z%3== 0 && 5*x + 3*y + z/3 == 100) printf("%d %d %d\n",
                    x,y,z);
            }
}

```

运行结果是：

0 25 75

4 18 78

8 11 81

12 4 84

**【例 9-4】** 下图为排球场地平面图，其中一、二、三、四、五、六为位置编号，二、三、四号位为前排，一、六、五号位为后排。某排球队开赛时，一、四号位放主攻手，二、五号位放二传手，三、六号位放副攻手，队员所穿球衣分别为1、2、3、4、5、6号，每个队员的球衣号与他们的站位号不同；已知1、6号队员不在后排；2、3号队员不是二传手；3、4号队员不在同一排；5、6号队员不是副攻手。试编写程序推算每个队员的站位情况。

四	三	二
五	六	一

算法：用穷举搜索法在每个位置上将每个队员都枚举排列一遍，看是否符合条件，

所以用六重循环来实现。循环次数为： $6^6=46656$ 。

```
#include<stdio.h>
void main()
{int a[6],i,j,k,flag,flag2;
for(a[0]=1; a[0]<=6; a[0]++)
for(a[1]=1; a[1]<=6; a[1]++)
for(a[2]=1; a[2]<=6; a[2]++)
for(a[3]=1; a[3]<=6; a[3]++)
for(a[4]=1; a[4]<=6; a[4]++)
for(a[5]=1; a[5]<=6; a[5]++)
{ flag = 0;
for (i=0; i<6; i++)
for (j=0; j<6; j++)
if (i != j)
if (a[i] == a[j]) /* 任意两个数都不应该相同 */
flag++;
if (flag == 0) /* 若是 6 个数的一个排列 */
{ flag2 = 0;
for (k=0; k<6; k++)
if (a[k] == k+1) /* 每个球员的球衣号与他们的站位不同 */
flag2++;
if (flag2 == 0)
if (a[0]!=1 && a[0]!=6 && a[4]!=1 && a[4]!=6 && a[5]!=1 && a[5]!=6)
/* 1,6 号队员不在后排 */
if (a[1]!=2 && a[1]!=3 && a[4]!=2 && a[4]!=3)
/* 2, 3 号队员不是二传手 */
if (a[2]!=5 && a[2]!=6 && a[5]!=5 && a[5]!=6)
/* 5,6 号队员不是副攻手 */
{ if (a[1]==3 || a[2]==3 || a[3]==3)
/* 3,4 号不在同一排 */
if ((a[0]==4 || a[4]==4 || a[5]==4) || (a[1]==4 || a[2]==4
|| a[3]==4))
printf("%d %d %d\n%d %d %d\n",a[3],a[2],a[1],a[4],
a[5], a[0]);
}
}
}
}
```

运行结果是：



3 1 6

4 2 5

### 9.2.3 递推法

【例 9-5】编写程序，用递推法计算菲波那（Fibonacci）级数的第  $n$  项。菲波那（Fibonacci）级数数列为：0, 1, 1, 2, 3, 5, 8, 13, ..., 即

$$F(0)=0, F(1)=1,$$

$$F(n)=F(n-1)+F(n-2) \quad \text{当 } n>1 \text{ 时。}$$

用递推法编写的程序为：

```
# include <stdio.h>
int f(int n)
{   int f0=0, f1=1, f, i;
    if (n==0) return 0;
    if (n==1) return 1;
    for (i=2; i<=n; i++)
    {   f = f0 + f1;    /* 由前两步的结果得到当前结果，“推出” */
        f0 = f1;       /* 把原来的前一步当作下一次的前两步，“传递” */
        f1 = f;        /* 把当前结果当作下一次的前一步，“传递” */
    }
    /* 在进行这种向前传递时，要注意传递的时序 */
    return f;
}

void main()
{   int n=7;
    printf("f(%d)=%d\n", n, f(n));
    n=30; printf("f(%d)=%d\n", n, f(n));
    n=40; printf("f(%d)=%d\n", n, f(n));
}
```

运行结果是：

f(7)=13

f(30)=832040

f(40)=102334155

上面的例子是当前项与前两项相关的问题，再给一个与前三项相关的例子。

【例 9-6】编写程序，用递推法计算广义菲波那（Fibonacci）级数的第  $n$  项。

广义菲波那（Fibonacci）级数数列为：0, 0, 1, 1, 2, 4, 7, 13, 24, ..., 即：

$$F(0)=0, F(1)=0, F(2)=1$$

$$F(n)=F(n-1)+F(n-2)+F(n-3) \quad \text{当 } n>2 \text{ 时。}$$

用递推法编写的程序为:

```
# include <stdio.h>
int f(int n)
{   int f0=0, f1=0, f2=1, f, i;
    if (n==0 || n==1) return 0;
    if (n==2) return 1;
    for (i=3; i<=n; i++)
    {   f = f0 + f1 + f2;          /* 由前三步的结果得到当前结果*/
        f0 = f1; f1 = f2; f2 = f; /* 在进行这种向前传递时, 要注意传递的时序 */
    }
    return f;
}

void main()
{   int n=8;
    printf("f(%d)=%d\n", n, f(n));
    n=10; printf("f(%d)=%d\n", n, f(n));
    n=20; printf("f(%d)=%d\n", n, f(n));
}
```

运行结果是:

f(8)=24

f(10)=81

f(20)=35890

上面的例子都是给出了数学表达的递推式, 下面再给一个例子, 分析如何自己推导出递推式。

【例 9-7】 计算下列级数和:

$$S(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

直到  $\frac{x^n}{n!} < 10^{-6}$ 。

分析: 设级数的第  $n$  项为:  $T_n = \frac{x^n}{n!}$ ,

则第  $n+1$  项:  $T_{n+1} = \frac{x^{n+1}}{(n+1)!} = \frac{x^n}{n!} \times \frac{x}{n+1} = T_n \times \frac{x}{n+1}$

得到递推式:  $T_{n+1} = T_n \times \frac{x}{n+1}$ ,  $T_0 = 1$ 。

由此编写的程序为:

```

#include <stdio.h>
#include <math.h>
double f(double x)
{   float s=1.0, t=1.0; /* S=T0, T0=1 */
    int n=1;
    do { t = t * x / n; /* Tn+1=Tn * x/(n+1) */
        s += t; n++;
    } while (fabs(t) > 1e-6);
    return s;
}
main()
{   printf("%12.6f %12.6f \n", f(2.0), exp(2.0)); /* ex */ }

```

运行结果是: 7.389057 7.389056

### 9.2.4 递归法

**【例 9-8】** 对于例 9-5 的非波那级数问题, 递归程序为:

```

int f(int n)
{   if (n== 0) return 0;
    else if (n== 1) return 1;
    else return f(n-1) + f(n-2);
}

```

**【例 9-9】** 编程求解 Hanoi 塔问题: 即有  $n$  个盘子在 A 处, 盘子从大到小, 最上面的盘子最小, 现在要把这  $n$  个盘子从 A 处搬到 C 处, 可以在 B 处暂存, 但任何时候不能出现大的压在小的上面的情况。

分析: (1) 若  $n=1$ , 则可以把盘子直接从 A 处搬到 C 处;

(2) 假设  $n-1$  时, 知道如何搬;

(3) 则  $n$  时, 根据 (2) 的假设, 可以先把前  $n-1$  个盘子从 A 处通过 C 处搬到 B 处, 就可以把第  $n$  个盘子直接从 A 处搬到 C 处, 再把前  $n-1$  个盘子从 B 处通过 A 处搬到 C 处, 则完成了全部盘子的搬动。

根据这个算法, 可以编写出如下的程序:

```

# include <stdio.h>
void move(int n, char a, char c)
{   static int Step=1;
    printf("Step %2d: Disk %d %c ---> %c\n", Step, n,a,c);
    Step++;
}

```

```

void Hanoi(int n, char a, char b, char c)
{
    if (n>1)
    {
        Hanoi(n-1, a, c, b);          /* 先将前 n-1 个盘子从 a 通过 c 搬到 b */
        move(n, a, c);                 /* 将第 n 个盘子从 a 搬到 c */
        Hanoi(n-1, b, a, c);          /* 再将这前 n-1 个盘子从 b 通过 a 搬到 c */
    }
    else move(n, a, c);                /* 将这 1 个盘子从 a 搬到 c */
}

void main()
{
    Hanoi(3, 'A', 'B', 'C');
}

```

运行结果为:

Step 1: Disk 1 A ---> C

Step 2: Disk 2 A ---> B

Step 3: Disk 1 C ---> B

Step 4: Disk 3 A ---> C

Step 5: Disk 1 B ---> A

Step 6: Disk 2 B ---> C

Step 7: Disk 1 A ---> C

**【例 9-10】** 给出从自然数 1, 2, 3, ..., n 中任取 k 个数的所有组合, 如 n=5, k=3。

这个组合问题是  $f(m, n)$  类型、即两个变量的问题, 可以这么想:

(1) 当  $k=1$  时, 即从 n 个数中取 1 个, 很显然, 是这 n 个数的每 1 个, 共 n 种取法;

(2) 当  $n=k$  时, 即从 k 个数中取 k 个, 很显然, 是这 k 个数的全部, 共 1 种取法;

(3) 假设已经知道如何从  $n-1$  个数中取 k 个的方法 (当然也知道从  $n-1$  个数中取  $k-1$  个的方法), 那么当从 n 个中取 k 个时, 可以归结为两种情况:

(a) 从前  $n-1$  个数中取 k 个。

(b) 先从前  $n-1$  个数中取  $k-1$  个, 再加上第 n 个数, 一共 k 个。

依据上述思路的递归程序是:

```

#include <stdio.h>
#define N 100
int a[N], nn;
void comb(int n, int k)
{
    void output();
    if (n > k && k > 1) /* 当 n 大于 k 并且 k 大于 1 时, 递归 */
    {
        comb(n-1, k); /* 在前 n-1 个中取 k 个元素的组合 */
        a[k] = n;      /* 先取第 n 个作为一个元素 */
        comb(n-1, k-1); /* 再在剩下的 n-1 个中取 k-1 个元素的组合 */
    }
}

```



```

    }          /* 递归后, 把 n 个元素取 k 个的组合问题, 变成了 n-1 个中取 k 个、
               n-1 个中取 k-1 个的组合问题 */
    else if (k == 1)          /* 若 k 为 1, 则分别取其中每一个元素作为一个组合*/
    {   for (; n>0; n--)
        {   a[k] = n;
            output(a, nn);
        }
    }
    else if (n==k)          /* 若 n 等于 k, 则将这 n 个元素作为一个组合*/
    {   for (; n>0; n--)
        {   a[n] = n;
            output(a, nn);
        }
    }
}

void output(int a[], int nn)
{   int i; static int count=1;
    printf("%3d: ", count++);
    for (i=1; i<=nn; i++) printf("%d ", a[i]);
    printf("\n");
}

void main()
{   comb(5,3); }

```

输出结果为:

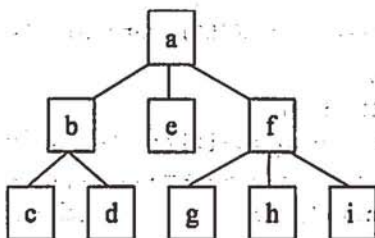
```

1: 1 2 3
2: 1 2 4
3: 2 3 4
4: 1 3 4
5: 1 2 5
6: 2 3 5
7: 1 3 5
8: 3 4 5
9: 2 4 5
10: 1 4 5

```

**【例 9-11】** 设 M 叉树采用列表法表示, 即每棵子树对应一个列表, 列表的结构为: 子树根结点的值后跟用“( )”括起来的各子树的列表(若有子树的话), 各子树的列表间用“,”分隔。例如下面的三叉树可用列表: a(b(c,d),e,f(g,h,i)) 表示。

编写程序根据输入的列表生产一棵 M 叉树, 并由 M 叉树再输出列表。



分析：生成 M 叉树时，在列表中遇到左括号 ‘(’ 后，表示是子树的开始，即可递归调用生成过程；输出 M 叉树时，遇到任何子树则递归调用输出过程。

```

#include <stdio.h>
#include <stdlib.h>
#define M 3 /* 三叉树 */
typedef struct node {
    int val;
    struct node *subTree[M];
} NODE;
char buf[255], *str=buf;
NODE *d = NULL;
NODE *makeTree() /* makeTree 由列表生产 M 叉树，并返回首指针 */
{
    int k; NODE *s;
    s = (NODE *)malloc(sizeof(NODE));
    s->val = *str++;
    for (k=0; k<M; k++) s->subTree[k] = NULL;
    if (*str == '(')
    {
        k=0;
        do {
            str++; /* 跳过 '(' 或 ',' */
            s->subTree[k] = makeTree(); /* 对子树开始递归 */
            if (*str == ')')
            {
                str++; /* 跳过 ')' */
                break; /* 当前子树结束 */
            }
            k=k+1;
        } while (*str);
    }
    return s;
}
void walkTree(NODE *t) /* walkTree 由 M 叉树输出列表 */
{
    int i;
    if (t != NULL)
    {
        putchar(t->val); /* 输出当前子树根结点的值 */

```

```

        if (t->subTree[0] == NULL) return; /* 表示当前 t 是叶子结点 */
        putchar('(');
        for (i=0; i<M; i++)
        {   walkTree(t->subTree[i]); /* 对每棵子树开始递归 */
            if (i != M-1 && t->subTree[i+1] != NULL)
                putchar(','); /* 不是最后一个树杈, 则输出 ',' */
        }
        putchar(')');
    }
}

void main()
{   printf("输入表达式: "); scanf("%s", str);
    d = makeTree(); walkTree(d); putchar('\n');
}

```

## 9.2.5 回溯法

**【例 9-12】** 求解迷宫问题。迷宫用一个二维数组表示，其元素值有两个：0 和 1，0 表示通路，1 表示阻塞，数组左上方[0][0]元素为迷宫入口，右下方[M-1][N-1]元素为迷宫出口，现在要求寻找一条从入口到出口的通路（并不一定是最短的）。

分析：迷宫中除了边沿的元素外，其他位置可以选择的移动方向有 8 个，首先沿着一个方向（从右开始）前进，若无路可走，则换一个方向试探，若某处 7 个方向（除了来时的方向）都不通，则开始回溯，后退一步继续上述过程，若已经退到入口处仍不通，则说明无解。

为了回溯的需要，用栈记录下走过的每一步的位置 (i, j) 和方向 v。最终给出的解的路径上的元素值置为 8，曾经到达过、但无路可走而被迫回溯的元素置为 4。

基于上述思想编写出的程序为：

```

#include <stdio.h>
#define M 12
#define N 15
void maze(int p[][N], int m, int n)
{   int is[8]={0,1,1,1,0,-1,-1,-1};
    /* 行前进方向: 右,右下,下,左下,左,左上,上,右上 */
    int js[8]={1,1,0,-1,-1,-1,0,1};
    /* 列前进方向: 右,右下,下,左下,左,左上,上,右上 */
    int stack[3*M*N], *top;
    int i,j,v,g,h,jt;
    top = stack;

```

```

i = j = v = 0;
if (p[0][0] != 0)      /* 入口无路可走, 提示无路, 返回 */
{
    p[0][0] = 4; printf("no path!\n"); return; }
while (top != stack || v != 7)
{
    g=i+is[v]; h=j+js[v]; jt=0;
    if (g>-1 && g<m && h>-1 && h<n)
    {
        if (g==m-1 && h==n-1 && p[m-1][n-1] == 0)
        {
            p[i][j] = 8; p[g][h] = 8;
            return;      /* 找到路径, 停止返回 */
        }
        if (p[g][h] == 0)
        {
            p[g][h]=4;
            p[i][j]=8; /* 前一步位置上置为 8 */
            top += 3;
            top[0]=i; top[1]=j; top[2]=v; /* 入栈, 记录下前一步 */
            i=g; j=h; v=0; jt=1;
        }
    }
    if (jt == 0)
    {
        if (v<7) v++; /* 换一个方向 */
        else
        {
            while (top != stack && top[2] == 7)
            {
                p[ top[0] ][ top[1] ] = 4;
                /* 无路, 栈中元素置为 4, 并退栈 */
                top -= 3;
            }
            if (top != stack)
            {
                i=top[0]; j=top[1]; v=top[2]; /* 回溯, 退栈 */
                p[i][j] = 4; top -= 3;
            }
        }
    }
}
printf("no path!\n"); /* 无路可走, 提示无路, 返回 */
return;
}

void main()
{
    int i,j;
    int p[M][N] = { {0,1,0,0,0,1,1,0,0,0,1,1,1,1,1},
                     {1,0,0,0,1,1,0,1,1,1,0,0,1,1,1},
                     {0,1,1,0,0,0,0,1,1,1,1,0,0,1,1},

```



```

        {1,1,0,1,1,1,1,0,1,1,0,1,1,0,0},
        {1,1,0,1,1,1,1,0,1,1,0,1,1,0,0},
        {1,1,0,1,0,0,1,0,1,1,1,1,1,1,1},
        {0,0,1,1,0,1,1,1,0,1,0,0,1,1,1},
        {0,1,1,1,1,0,0,1,1,1,1,1,1,1,1},
        {0,0,1,1,0,1,1,0,1,1,1,1,1,0,1},
        {1,1,0,0,0,1,1,0,1,1,0,0,0,0,0},
        {0,0,1,1,1,1,1,0,0,0,1,1,1,1,0},
        {0,1,0,0,1,1,1,1,1,0,1,1,1,1,0}};

    maze(p, 12, 15);
    for(i=0; i<M; i++)
    {   for (j=0; j<N; j++) printf("%d ", p[i][j]);
        printf("\n");
    }
}

```

运行结果为:

```

810001144411111
188811411144111
011884411114411
118111141141144
118111141141144
118100141111111
081101114100111
811118811111111
081181181111101
118881181188888
001111108811118
010011111411118

```

## 9.2.6 贪婪法

**【例 9-13】** 编写程序, 实现人民币的找零: 任何一张面额的纸币都找成比它的面额小的纸币, 最小单位是元。

**分析:** 由于人民币的面额有 100 元、50 元、20 元、10 元、5 元、2 元、1 元, 任何一张较大面额的纸币, 都采用贪婪法首先分解成尽可能大的小额纸币, 比如, 50 元分解时, 首先分解出一个 20 元, 在剩余的部分 30 元中, 再分解出一个 20 元, 剩余部分 10 元可以用 10 元的纸币, 因此分解结果是:  $50=20+20+10$ 。

依此思想编写程序为:

```
# include <stdio.h>
# define N 20
int find(int n, int m, int *d, int c, int *pd)
{
    int r;
    if (n==0) return 0; /* 已分解完成 */
    if (m==0 || c==0) return -1; /* 不可以分解 */
    if (n < *d) return find(n, m, d+1, c-1, pd);
    else
    {
        *pd = *d; /* 按能找零的最大数给 */
        r = find(n-*d, m-1, d, c, pd+1); /* 继续对剩余部分作分解 */
        if (r >= 0) return r+1;
        return -1; /* 不可以分解 */
    }
}

void main()
{
    int n,m, k, i, p[N];
    int d[7]={100, 50, 20, 10, 5, 2, 1};
    printf("请输入钱数和找零最大张数 n,m: ");
    scanf("%d%d", &n, &m);
    for (i=0; i<7; i++)
        if (n > d[i]) break; /* 跳过面额比它大或相等的纸币单位 */
    k = find(n, m, &d[i], 7-i, p);
    if (k <= 0) printf("找不开!\n");
    else
    {
        printf("%d=%d", n, p[0]);
        for (i=1; i<k; i++) printf("+%d", p[i]);
        printf("\n");
    }
}
```

运行结果为:

请输入钱数和找零最大张数 n,m: 50 5

50=20+20+10

请输入钱数和找零最大张数 n,m: 100 5

100=50+50

## 9.2.7 分治法

【例 9-14】 编写程序用快速排序法将无序的整数序列从小到大排序。

分析：快速排序法的基本思想就是分治法，通过一次分割，将无序序列分成两部分，其中前一部分的元素值均小于后一部分的元素值。然后每一部分再各自递归进行上述过程，直到每一部分的长度为 1 为止。我们知道排序的计算量通常是与序列长度的平方成正比，所以分治法将序列一分为二再各自排序，将大大降低计算量，所以快速排序是一种高效的排序方法。

对序列的分隔过程为：首先，在序列的第一个、中间一个以及最后一个元素中选取中项，设为  $p[l]$ ，并将  $p[l]$  赋给  $t$ ，再将序列中的第一个元素移到  $p[l]$  的位置上。然后设置两个指针  $i$ 、 $j$  分别指向序列的起始和最后的位置。重复以下两步，直到  $i=j$  为止，此时将  $t$  移到  $p[i]$  的位置上：

(1)  $j$  逐步减小，直到发现一个  $p[j]<t$  为止，将  $p[j]$  移到  $p[i]$  的位置上；

(2)  $i$  逐步增大，直到发现一个  $p[i]>t$  为止，将  $p[i]$  移到  $p[j]$  的位置上。

依此思想编写出的程序为：

```
# include <stdio.h>
void split(int p[], int n, int *m)
{
    int i, j, k, l, t;
    i=0;    j=n-1; k=(i+j)/2;
    /* 在 i, j, k 元素中选取中项 */
    if ( p[i] >= p[j] && p[j] >= p[k] ) l = j;
    else if ( p[i] >= p[k] && p[k] >= p[j] ) l = k;
    else    l = i;
    t = p[l]; p[l] = p[i];
    while (i != j)
    {
        while (i < j && p[j] >= t) j--; /* j 逐步减小，直到发现一个 p[j] < t 为止 */
        if (i < j)
        {
            p[i] = p[j]; i++;
            while (i < j && p[i] <= t) i++;
            /* i 逐步增大，直到发现一个 p[i] > t 为止 */
            if (i < j) { p[j] = p[i]; j--; }
        }
    }
    p[i] = t;    *m = i;
    return;
}

void qsort(int p[], int n)
{
    int i;
    if (n > 1)
    {
        split(p, n, &i); /* 分隔，找到分治点 i，i 元素是不动点，不再参加排序，
                           是此元素在最终有序序列中的位置 */
    }
}
```

```

    qsort(p, i);    /* 将序列的前半部分 0~i-1 进行排序 */
    qsort(&p[i+1], n-i-1); /* 将序列的后半部分 i+1~n-1 进行排序 */
}
return ;
}

void main()
{
    int i, a[20]={1, 9, 3, 7, 18, 2, 20, 4, 16, 5, 15, 6, 14, 7, 13, 6,
    12, 8, 11, 10};
    qsort(a, 20);
    for (i=0; i<20; i++)    printf("%d ", a[i]);
}

```

运行结果为: 1 2 3 4 5 6 6 7 7 8 9 10 11 12 13 14 15 16 18 20

## 9.2.8 动态规划法

**【例 9-15】** 在实际的工作中, 进行汉字的印刷体识别或语音识别时, 需要比较识别结果的汉字串的识别字正确率、插入错误率、删除错误率以及替代错误率, 请编写程序实现。

方法是: 把识别出来的汉字串与标准汉字串进行匹配, 也就是求两个字符串序列的最长公共子序列的长度。采用动态规划法, 将求最长公共子序列问题分解成子问题: 设  $A = "a_0 a_1, \dots, a_{m-1}"$ ,  $B = "b_0 b_1, \dots, b_{n-1}"$ , 并设  $Z = "z_0 z_1, \dots, z_{k-1}"$  为它们的最长公共子序列, 则有:

(1) 若  $a_{m-1} = b_{n-1}$ , 则  $z_{k-1} = a_{m-1} = b_{n-1}$ , 且  $"z_0 z_1, \dots, z_{k-2}"$  是  $"a_0 a_1, \dots, a_{m-2}"$  和  $"b_0 b_1, \dots, b_{n-2}"$  的一个最长公共子序列;

(2) 若  $a_{m-1} \neq b_{n-1}$ , 则若  $z_{k-1} \neq a_{m-1}$ , 蕴涵  $"z_0 z_1, \dots, z_{k-1}"$  是  $"a_0 a_1, \dots, a_{m-2}"$  和  $"b_0 b_1, \dots, b_{n-1}"$  的一个最长公共子序列;

(3) 若  $a_{m-1} \neq b_{n-1}$ , 则若  $z_{k-1} \neq b_{n-1}$ , 蕴涵  $"z_0 z_1, \dots, z_{k-1}"$  是  $"a_0 a_1, \dots, a_{m-1}"$  和  $"b_0 b_1, \dots, b_{n-2}"$  的一个最长公共子序列。

因此, 在找 A 和 B 的公共子序列时, 若有  $a_{m-1} = b_{n-1}$ , 则只需进一步解决子问题: 找  $"a_0 a_1, \dots, a_{m-2}"$  和  $"b_0 b_1, \dots, b_{n-2}"$  的最长公共子序列; 若有  $a_{m-1} \neq b_{n-1}$ , 则需要解决两个子问题: 找  $"a_0 a_1, \dots, a_{m-2}"$  和  $"b_0 b_1, \dots, b_{n-1}"$  的最长公共子序列, 和找  $"a_0 a_1, \dots, a_{m-1}"$  和  $"b_0 b_1, \dots, b_{n-2}"$  的最长公共子序列, 并从两个结果中取较长的一个作为 A 和 B 的最长公共子序列。

定义  $c[i][j]$  为序列  $"a_0 a_1, \dots, a_{m-1}"$  和  $"b_0 b_1, \dots, b_{n-1}"$  的最长公共子序列的长度,  $c[i][j]$  的求解可以递归表述为:

(a)  $c[i][j] = 0$ , 若  $i=0$ , 或  $j=0$ ;

(b)  $c[i][j] = c[i-1][j-1] + 1$ , 若  $i, j > 0$ , 且  $a[i-1] = b[j-1]$ ;



(c)  $c[i][j] = \max(c[i][j-1], c[i-1][j])$ , 若  $i, j > 0$ , 且  $a[i-1] \neq b[j-1]$ 。

按此算法可以求出两个字符串序列的最长公共子序列, 在最长公共子序列 (即正确汉字数) 求得以后, 利用两个字符串序列的匹配点, 就可以进一步计算出字符串的插入错误率、删除错误率以及替代错误率, 后者不涉及动态规划方法, 因此这里不再叙述其算法, 仅给出程序。

依此思想编写出的程序为:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
long corword, allword, all_ins_err, all_del_err, all_sub_err, Total;
char rmatrix[100][100];
char rjump[100][100];
void compare(short *, int, short *, int);
/* 将两个字的一个汉字按其内码转化为一个短整数, 便于下面的比较 */
void Char2Short(unsigned char *str, short *num, int length)
{
    int i;
    for (i=0; i<length; i++) num[i] = (str[2*i] - 0xA0)*94 + str[2*i+1]
        - 0xA0;
}
void main(char argc, char* argv[])
{
    FILE *fp1, *fp2;
    int i, RightSentence;
    float Right, InsErr, DelErr, SubErr;
    unsigned char str1[201], str2[201];
    short Want2Search[100], Result[100];
    int WantLength, ResultLength;
    if (argc < 3)
    {
        printf("Usage: WordRight <rightfilename> <wrongfilename>\n");
        exit(0);
    }
    if ( (fp1 = fopen(argv[1], "r")) == NULL)
    {
        printf("Can't open file: %s\n", argv[1]); exit(0);
    }
    if ( (fp2 = fopen(argv[2], "r")) == NULL)
    {
        printf("Can't open file: %s\n", argv[2]); exit(0);
    }
    corword=allword=all_ins_err=all_del_err=all_sub_err=i=0;
    while(1)
    {
        if(fgets(str1, 200, fp1) == NULL || fgets(str2, 200, fp2) == NULL)
            break;
        ResultLength = strlen(str1)/2;
        Char2Short(str1, Result, ResultLength); /* 将汉字转换成整数 */
        WantLength = strlen(str2)/2;
```

```

    Char2Short(str2, Want2Search, WantLength); /* 将汉字转换成整数 */
    compare(Result, ResultLength, Want2Search, WantLength);
    if (!strcmp(str1, str2)) /* 两个句子完全相同 */
        RightSentence++;
    i++;
}
fclose(fp1); fclose(fp2);
Right=(float)corword/(allword) *100;
Total = corword + all_sub_err + all_ins_err + all_del_err;
InsErr=(float)all_ins_err/Total *100;
DelErr=(float)all_del_err/Total *100;
SubErr=(float)all_sub_err/Total *100;
printf("SentenceNum=%d Right=%6.2f%% InsErr=%6.2f%% DelErr=%6.2f%%
        SubErr=%6.2f%% SenRight=%6.2f%%\n",
        Right, InsErr, DelErr, SubErr, 100.0*RightSentence/i);
}

void compare(short *str1, int right_len, short *str2, int wrong_len)
{
    register short i,j; char score; char wrongtag[100];
    short inode,before,ins_err,del_err,sub_err,period,right_period;
    for(i=0; i<wrong_len; i++)
        for(j=0; j<right_len; j++) rmatrix[i][j]=0;
    for(j=0; j<right_len; j++) /* initial first row of rmatrix */
        if (str2[0] == str1[j]) rmatrix[0][j]=1;
    for(i=1; i<wrong_len; i++)
    {
        if(str2[i] == str1[0]) rmatrix[i][0]=1; /*initial first collum
        of rmatrix */
        for(j=1; j<right_len; j++)
        {
            score=0;
            if (str2[i] == str1[j]) score=1; /* 对应字相等, 则得分为 1 */
            score += rmatrix[i-1][j-1]; /* 将当前得分累计以前的得分 */
            /* 将(rmatrix[i-1][j], rmatrix[i][j-1], score)中最大的一个赋给
            rmatrix[i][j] */
            if(score >= rmatrix[i-1][j] && score >= rmatrix[i][j-1])
            {
                rmatrix[i][j]=score; rjump[i][j]=0;
            }
            else if(rmatrix[i][j-1] > rmatrix[i-1][j])
            {
                rmatrix[i][j] = rmatrix[i][j-1]; rjump[i][j]=1;
            }
            else
            {
                rmatrix[i][j] = rmatrix[i-1][j]; rjump[i][j]=2;
            }
        }
    }
}

for(i=0; i<wrong_len; i++) wrongtag[i]=100;

```

```

corword += rmatrix[wrong_len-1][right_len-1]; /* 正确字数 */
i=wrong_len-1; j=right_len-1;
while(i>0)
{ score = rjump[i][j];
  switch(score)
  { case 0:
    if(rmatrix[i][j]>rmatrix[i-1][j-1]) wrongtag[i] =(char)j;
    i--; j--;
    break;
    case 1: j--; break;
    case 2: i--; break;
  }
}
ins_err=del_err=sub_err=0 ;
inode=0 ;
while(wrongtag[inode]==100 && inode < wrong_len) inode++ ;
if(inode==wrong_len)
{ period=wrong_len ;
  right_period=right_len ;
  if((period>right_period) )
  { ins_err += period - right_period ; sub_err += right_period ; }
  else if(period<right_period)
  { del_err += right_period - period ; sub_err += period ; }
  else if(period!=1) sub_err += right_period ;
}
else
{ period=inode+1 ;
  right_period=wrongtag[inode]+1 ;
  if(period>right_period)
  { ins_err += period-right_period ; sub_err += right_period ; }
  else if(period<right_period)
  { del_err += right_period-period ; sub_err += period ; }
  else if (period!=1) sub_err += right_period ;
}
before=inode ;
inode++ ;
while(inode<wrong_len)
{ if(wrongtag[inode]!=100)
{ period=inode-before ;
  right_period=wrongtag[inode]-wrongtag[before] ;
  if((period>right_period) )
  { ins_err += period - right_period ;

```

```

        sub_err += right_period;
    }
    else if(period < right_period)
    { del_err += right_period - period ;
      sub_err += period ;
    }
    else if(period!=1)
      sub_err += right_period ;

    if(rmatrix[inode][wrongtag[inode]]==corword)
    {   period = wrong_len - (inode+1) ;
        right_period = right_len - (wrongtag[inode]+1) ;
        if(period > right_period)
        {   ins_err += period - right_period ; sub_err += right_
            period; }
        else if(period < right_period)
        {   del_err += right_period - period ; sub_err += period ; }
        else if(period != 1) sub_err += right_period ;
        break ;
    }
    before=inode ;
}
inode++ ;
}
all_ins_err += ins_err ;      /* 统计总的插入错误字数 */
all_del_err += del_err ;      /* 统计总的删除错误字数 */
all_sub_err += sub_err ;      /* 统计总的替代错误字数 */
allword += right_len ;        /* 统计总的字数 */
}

```

### 9.3 思考练习题及答案

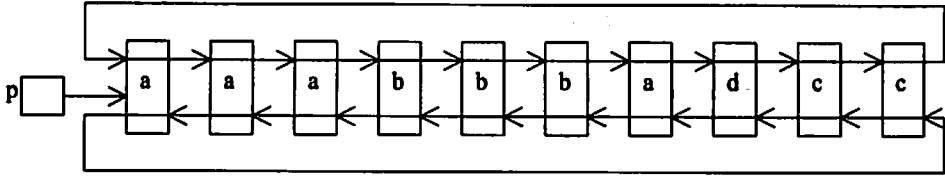
#### 思考练习题

1. 填空题：设一个环上有编号  $0 \sim n-1$  的  $n$  粒不同颜色的珠子（每粒珠子颜色用字母表示， $n$  粒珠子的颜色由输入的字符串表示）。以环上某两粒珠子间为断点，从断点一方按顺时针方向取走连续同色的珠子，又从断点另一方按逆时针方向对剩下珠子取走连续同色的珠子，两者之和为该断点可取走珠子的粒数。移动断点，能取走的珠子数不尽相同。



本程序找出可以取走最多的珠子颜色及断点的位置。程序中用双向链表存储字符串。

例如：编号为 0~9 的 10 粒珠子颜色的字符串为 “aaabbbadcc”，对应链表为下图，若在 2 号和 3 号珠子间为断点，共可取走 6 粒珠子，且能取走的珠子数最多。



```
# include <stdio.h>
# include <string.h>
# include <malloc.h>
typedef struct node {
    char ch;                /* 珠子颜色 */
    struct node *fpt;        /* 后继指针 */
    struct node *bpt;        /* 前趋指针 */
} NODE;
NODE *building(char *s)     /* 生成双向循环链表 */
{
    NODE *p = NULL, *q;
    while (*s)
    {
        q = (NODE *)malloc(sizeof(NODE));
        q->ch = *s++;
        if (p == NULL) p = q->fpt = q->bpt = q;
        else
        {
            p->bpt->fpt = q; q->fpt=p;
            q->bpt=__ (1) __;
            __ (2) __;
        }
    }
    return p;
}

int count(NODE *start, int maxn, int step) /* 求可取走珠子粒数 */
{
    int color, c; NODE *p;
    color = -1; c = 0;
    for (p=start; c<maxn; p=step>0 ? p->fpt : p->bpt) /* */
    {
        if (color == -1) color = p->ch;
        else if (__ (3) __) break;
        c++;
    }
    return c;
}
```

```

}
int find(char *s, int *cutpos) /* 寻找取走珠子数最多的断点和粒数 */
{
    int i, c, cut, maxc=0, len=strlen(s);
    NODE *p;
    if ((p = building(s)) == NULL) { *cutpos = -1; return -1; }
    i = 0;
    do {
        c = count(p, len, 1); /* 从端点指针开始向后统计相同珠子个数 */
        c = c + (4);
        if (c>maxc) { maxc=c; cut = i; }
        (5);
        i++;
    } while (i<len);
    if (maxc > len)
        /* 若这串珠子就一种颜色, 上述算法会把珠子数重复统计两次, */
maxc = len; /* 所以以最大长度为准 */
    *cutpos = cut;
    return maxc;
}
void main()
{
    int cut, max; char s[120]="aaabbbbadcc";
    max = find(s, &cut);
    printf("Cut position=%d, Number=%d\n", cut, max);
}

```

2. 填空题: 本程序采用递归算法将一个自然数  $n$  分解成不多于  $m$  个整数之和。设构成和数  $n$  的各个整数均取自于数组  $d$ ,  $d$  中的整数互不相等且由大到小存储。

```

#include <stdio.h>
#define N 20
int find(int n, int m, int *d, int c, int *pd)
{
    int r;
    if (n==0) return 0; /* 已分解完成 */
    if (m==0 || c==0) return -1; /* 不可以分解 */
    if ((1))
        return find(n, m, d+1, c-1, pd);
    else
    {
        *pd = *d;
        r = find((2), d, c, (3)); /* 继续对剩余数作分解 */
        if (r >= 0) return (4);
        return find(n, m, (5), pd);
    }
}

```

```

    }
}

void main()
{
    int n,m, k, i, p[N], *pptr=p;
    int d[]={100, 81, 64, 49, 36, 25, 16, 9, 4, 1};
    printf("Enter n,m:"); scanf("%d%d", &n, &m);
    k = find(n, m, d, 10, pptr);
    if (k <= 0) printf("No answer!\n");
    else
    {
        printf("%d=%d", n, p[0]);
        for (i=1; i<k; i++) printf("+%d", p[i]);
        printf("\n");
    }
}

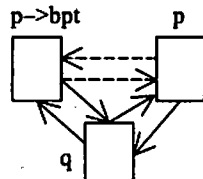
```

## 思考练习题答案

1. (1)  $p \rightarrow bpt$

(2)  $p \rightarrow bpt = q$

分析: building 的功能是生成双向链表, 而双向链表插入一个结点, 需要有四个指针赋值:  $p$  的前趋结点的后继指针指向  $q$  (即  $p \rightarrow bpt \rightarrow fpt = q$ ),  $q$  的后继指针指向  $p$  (即  $q \rightarrow fpt = p$ ),  $q$  的前趋指针指向  $p$  的前趋指针 ( $q \rightarrow bpt = p \rightarrow bpt$ ),  $p$  的前趋指针指向  $q$  (即  $p \rightarrow bpt = q$ ).



(3)  $p \rightarrow ch \neq color$

分析: count 的功能是从某个位置开始向前 (或向后) 统计相同颜色的珠子个数, 程序中填空处的前一个语句 `if (color == -1) color = p->ch;` 是取得开始处珠子的颜色, 而填空处是一个条件判断, 当条件为真时终止 (break), 即出现的珠子颜色不是 color 时终止, 因此应该填入  $p \rightarrow ch \neq color$ 。

(4)  $count(p \rightarrow bpt, len, -1)$

分析: find 的功能是寻找取走珠子数最多的断点和粒数, 是采用穷举搜索法, 将循环链表的任意一个位置都作为断点进行计数, 从而确定最多粒数的位置。填空处前一个语句 `c = count(p, len, 1);` 是从断点开始向后统计与断点处珠子颜色 ( $p \rightarrow ch$ ) 相同的珠子的个数, 因此本语句应该从断点的前一个位置 ( $p \rightarrow bpt$ ) 开始向前统计与断点处珠子颜色 ( $p \rightarrow bpt \rightarrow ch$ ) 相同的珠子的个数。而 count 函数中 step 是用来控制是向后还是向前搜索的开关, 当  $step > 0$  时向后, 而  $step \leq 0$  时向前, 因此此处只要填入一个小于等于 0 的数即可。因为此处答案为:  $count(p \rightarrow bpt, len, -1)$ 。

(5)  $p = p \rightarrow fpt$

分析: 由于是采用穷举搜索法, 此处是要换下一个断点位置继续搜索, 向前向后前

进一步都可以。所以此处可以填入:  $p = p \rightarrow \text{fpt}$  或  $p = p \rightarrow \text{bpt}$ 。

本程序的正确运行结果为: Cut position=3, Number=6

2. (1)  $n < *d$

分析: 本程序是采用递归法将一个自然数  $n$  分解成不多于  $m$  个指定的  $d$  序列整数之和。由于此处是逻辑判断, 当条件为真时返回:  $\text{find}(n, m, d+1, c-1, \text{pd})$ , 即将  $n$  分解成不多于  $m$  个  $d$  序列中跳过第一个整数的 ( $c-1$  个整数) 序列之和, 而跳过第一个整数的原因显然是由于  $n$  小于这个整数, 所以此处应该填入:  $n < *d$ 。

(2)  $n-*d, m-1$

(3)  $\text{pd}+1$

分析: 由于这两处填空都是函数  $\text{find}$  的参数, 所处的  $\text{else}$  是  $n \geq *d$ , 因此是先将  $*d$  作为  $n$  分解后的一个数放入  $\text{pd}$  数组中 ( $*\text{pd} = *d$ ), 然后将剩余部分  $n-*d$  再递归调用  $\text{find}$  分解成不多于  $m-1$  数的和, 因此填空 (2) 应该填入:  $n-*d, m-1$ 。而由于已经将  $n$  分解出一个整数放入了  $\text{pd}$  中, 所以下一个分解出的数应该放入  $\text{pd}$  的下一个单元中, 因此填空 (3) 应该填入:  $\text{pd}+1$ 。

(4)  $r+1$

分析: 由于  $r$  是  $\text{find}$  的返回值, 表示  $n-*d$  被分解成的整数个数, 连同事先分解出的  $*d$ , 共  $r+1$  个整数, 所以此处应该填入:  $r+1$ 。

(5)  $d, 0$

分析: 由于此处填空是函数  $\text{find}$  的参数, 而本  $\text{find}$  前是  $\text{return}$  语句, 即本次函数调用的结束。本  $\text{return}$  语句能被执行到, 是因为前面的  $\text{if}$  语句的条件为假, 没有执行前一个  $\text{return}$  语句, 也就是说  $r < 0$ , 说明  $n-*d$  不能分解成不多于  $m$  个整数之和 (不可以分解)。但填空处又是  $\text{find}$  函数递归调用, 说明此次函数调用不是为了整数分解, 应该是要结束分解过程, 而在  $\text{find}$  函数的第二个  $\text{if}$  语句:  $\text{if}(m==0 \parallel c==0) \text{return } -1$ ; 后的注释是: 不可以分解, 就是当  $m$  为 0 或  $c$  为 0 时, 认为是不可以分解。因此填空处  $c$  参数位置应该为 0,  $d$  参数处任意。所以此处应该填入:  $d, 0$

本程序的正确运行结果为:

Enter n,m: 100 2

100=100

Enter n,m: 13 2

13=9+4

Enter n,m: 14 2

No answer!

Enter n,m: 71 5

71=64+4+1+1+1



## 第 10 章 标准化基础知识

### 10.1 内容提要

#### 10.1.1 标准化的基本概念

##### 1. 标准、标准化的概念

标准是对重复性事物和概念所做的统一规定。标准以科学、技术和实践经验的综合成果为基础，以获得最佳秩序和促进最佳社会效益为目的，经有关方面协商一致，由主管或公认机构批准，并以规则、指南或特性的文件形式发布，作为共同遵守的准则和依据。规范、规程都是标准的一种形式。

标准化是在经济、技术、科学及管理等社会实践中，对重复性事物和概念通过制定、发布和实施标准，达到统一，获最佳秩序和社会效益的过程。

##### 2. 标准化的范围和对象

标准化涉及的范围：

- (1) 生产、经济、技术、科学及管理等社会实践中具有的重复性事物和概念；
- (2) 需要建立统一技术要求的各个领域。凡具有多次重复使用和需要制定标准的具体产品，以及各种定额、规划、要求、方法、概念等，都可称作为标准化的对象。

标准化对象一般可分为两大类：

- (1) 一类是标准化的具体对象，即需要制定标准的具体事物；
- (2) 另一类是标准化总体对象，即各种具体对象的全体所构成的整体，通过它可以研究各种具体对象的共同属性、本质和普遍规律。

##### 3. 标准化的实质

标准化的实质是通过制定、发布和实施标准，达到统一。

##### 4. 标准化的目的

- (1) 建立最佳秩序，即建立一定环境和一定条件的最合理秩序。
- (2) 获得最佳效益。

#### 10.1.2 标准化过程模式

标准是标准化活动的产物，其目的和作用，都是通过制定和贯彻具体的标准来体现的。标准化不是一个孤立的事物，而是一个活动过程。一个过程可分为几个子过程。

标准化活动过程，一般包括标准产生（调查、研究、形成草案、批准发布）子过程：

标准实施（宣传、普及、监督、咨询）子过程和标准更新（复审、废止或修订）子过程等。

### 1. 标准的制定

标准的产生一般包括：调查研究、制定计划（立项）、起草标准、征求意见、审查、批准发布等标准生成阶段。

ISO 和 IEC 是两个国际标准化组织，为规范国际标准的产生过程，发布了导则性文件。我国依据该导则性文件制定了相应的国家标准，以规范国家标准的制定程序。

### 2. 标准的实施

标准的实施过程，实际上就是推广和普及已被规范化的实践经验的过程。标准的实施过程包括哪些活动内容，没有统一的规定，一般有标准的宣传、贯彻执行和监督检查等。

（1）我国强制性标准的实施是通过强制性的监督检查来推动，依法开展标准的实施与监督。

（2）对于推荐性标准的实施，尚无明确有效的措施，需要建立和完善社会主义市场经济体制下的技术法规。

（3）《标准化法》、《国家标准管理办法》等法规，规定了我国标准化工作的方针、政策、任务和标准化体制等，是推行、实施标准化管理和监督的重要依据。

### 3. 标准的更新

标准的更新是实践经验的深化和提高的过程。人类社会实践是一个永无止息的活动，标准化也是一个永无止境的过程。

#### （1）标准复审

自标准实施之日起，至标准复审重新确认、修订或废止的时间，称为标准的有效期（也称为标龄）。标准有效期各个国家是不同。ISO 标准每 5 年复审一次，平均标龄为 4.92 年；1988 年发布的《中华人民共和国标准化法实施条例》中规定，标准实施后的复审周期一般不超过 5 年，即我国的国家标准有效期一般为 5 年。

#### （2）标准确认

经复审后的标准，若其内容仍符合当前科学技术水平并适合经济建设的需要，无需修改或只需做编辑性修改，可确认为继续有效。经确认的标准，在发布时，不改变标准的顺序号和年号。当标准再版时在标准封面写明 XXXX 年确认字样。

经过复审后的标准，若其内容需完善和补充，只要对标准条文、图、表作少量修改补充，仍可继续使用的，则采用标准修改单的形式，对需要修改补充的内容予以完善后，按照标准的制定程序，经标准化主管单位批准发布。

#### （3）标准修订

经复审后的标准，若其内容需要做较大修改才能适应生产和使用的需要以及科学技术发展的需要，则应作为修订项目。标准修订的程序按制定标准的程序执行。修订后的

标准顺序号不变, 年号改为重新修订发布时的年号。

### 10.1.3 标准的分类

标准化工作是一项复杂的系统工程, 标准为适应不同的要求从而构成一个庞大而复杂的系统, 为便于研究和应用, 可以从不同的角度和属性将标准进行分类。

#### 1. 根据适用范围分类

根据标准制定的机构和标准适用的范围, 可分为国际标准、国家标准、行业标准、企业(机构)标准及项目(课题)标准。

##### (1) 国际标准

国际标准是指国际标准化组织(ISO)、国际电工委员会(IEC)所制定的标准, 以及 ISO 出版的《国际标准题内关键词索引(KWIC Index)》中收录的其他国际组织制定的标准。国际标准在世界范围内统一使用, 各国可以自愿采用, 不强制适用。

##### (2) 国家标准

国家标准是由政府或国家级的机构制定或批准的、适用于全国范围的标准, 是一个国家的标准体系的主体和基础, 国内各级标准必须服从且不得与之相抵触。

##### (3) 区域标准

区域标准(也称地区标准)泛指世界上按地理、经济或政治划分的某一区域标准化团体所通过的标准。地区标准主要是指太平洋地区标准会议(PASC)、欧洲标准化委员会(CEN)、亚洲标准咨询委员会(ASAC)、非洲地区标准化组织(ARSO)等地区组织所制定和使用的标准。

##### (4) 行业标准

由行业机构、学术团体或国防机构制定, 并适用于某个业务领域的标准。如: (1) 美国电气和电子工程师学会标准 IEEE。(2) 中华人民共和国国家军用标准 GJB(由国防科学技术工业委员会批准, 适用于国防部门和军队使用的标准)。

##### (5) 企业标准(Company Standard)

由企业或公司批准、发布的标准, 某些产品标准由其上级主管机构批准、发布。

##### (6) 项目规范(Project Specification)

由某一科研生产项目组织制定, 且为该项任务专用的软件工程规范。

##### (7) 我国标准分类

根据《中华人民共和国标准化法》的规定, 我国标准分为国家标准、行业标准、地方标准和企业标准等四类。这四类标准主要是适用的范围不同, 不代表标准技术水平的高低。

(1) 国家标准: 由国务院标准化行政主管部门制定的需要全国范围内统一的技术要求, 称为国家标准。

(2) 行业标准: 没有国家标准而又需在全国某个行业范围内统一的技术标准, 由国



务院有关行政主管部门制定并报国务院标准化行政主管部门备案的标准,称为行业标准。

(3) 地方标准:没有国家标准和行业标准而又需在省、自治区、直辖市范围内统一的工业产品的安全、卫生要求,由省、自治区、直辖市标准化行政主管部门制定并报国务院标准化行政主管部门和国务院有关行业行政主管部门备案的标准,称为地方标准。

(4) 企业标准:企业生产的产品没有国家标准、行业标准和地方标准,由企业自行组织制定、作为组织生产依据的相应标准,或者在企业内制定适用的、比国家标准、行业标准或地方标准更严格的企业(内控)标准,并按省、自治区、直辖市人民政府的规定备案的标准(不含内控标准),称为企业标准。

## 2. 根据标准的性质分类

根据标准的性质可分为技术标准、管理标准、工作标准。

### (1) 技术标准

技术标准是针对重复性的技术事项而制定的标准,是从事生产、建设及商品流通时需要共同遵守的一种技术依据。(1)按其标准化对象特征和作用,可分为基础标准、产品标准、方法标准、安全卫生与环境保护标准等;(2)按其标准化对象在生产流程中的作用,可为零部件标准、工装标准、设备维修保养标准及检查标准等;(3)按标准的强制程度,可分为强制性与推荐性标准;(4)按标准在企业中的适用范围,又可分为公司标准、工厂标准和科室标准等。

### (2) 管理标准

管理标准是管理机构为行使其管理职能而制定的具有特定管理功能的标准。在实际工作中通常按照标准所起的作用不同,将管理标准分为技术管理标准、生产组织标准、经济管理标准、行政管理标准、业务管理标准和工作标准等。

### (3) 工作标准

对工作的内容、方法、程序和质量要求所制定的标准,称为工作标准。对生产和业务处理的先后顺序、内容和要达到的要求所作的规定称为工作程序标准。以管理工作为对象所制定的标准,称为管理工作标准。

## 3. 根据标准化的对象和作用分类

根据标准的对象和作用,标准可分为基础标准、产品标准、方法标准、安全标准、卫生标准、环境保护标准、服务标准等。

### (1) 基础标准

在一定范围内作为其他标准的基础并普遍通用,具有广泛指导意义的标准。

### (2) 产品标准

为保证产品的适用性,对产品必须达到的某些或全部特性要求所制定的标准。产品标准是一定时期和一定范围内具有约束力的产品技术准则;是产品生产、质量检验、选购验收、使用维护和洽谈贸易的技术依据。

### (3) 方法标准



以各种方法为对象而制定的标准。方法标准一般包括两类，一类以试验、检查、分析、抽样、统计、计算、测定、作业等方法为对象制定的标准。另一类是为合理生产优质产品，并在生产、作业、试验、业务处理等方面为提高效率而制定的标准。

#### (4) 安全标准

以保护人的安全或物品的安全为对象和目的而制定的标准。安全标准一般有两种形式：一种为专门目的安全标准；另一种是在产品标准或工艺标准中列出有关安全的要求和指标。

#### (5) 卫生标准

为保护人的健康，对食品、医药及其他方面的卫生要求而制定的标准。卫生标准是专门以卫生（工业卫生、劳动卫生等）要求为对象、目的制定的标准，如食品卫生标准，规定了食品中有害物质或病菌的限量。

#### (6) 环境保护标准

为保护环境不受污染和有利于生态平衡，对大气、水体、土壤、噪声、振动、电磁波等环境质量、污染管理、监测方法及其他事项而制定的标准。例如，水质标准、噪声标准等。

#### (7) 服务标准

为提高服务质量，对某项服务工作要达到的要求所制定的标准。服务包括技术服务即产品售后服务，也包括第三产业即旅游、商业、交通运输、金融、电讯、信息、保险、医疗等服务行业。

### 4. 根据法律的约束性分类

根据标准的法律约束性，可分为强制性标准、推荐性标准。

#### (1) 强制性标准

根据《标准化法》的规定，企业和有关部门对涉及其经营、生产、服务、管理有关的强制性标准都必须严格执行，任何单位和个人不得擅自更改或降低标准。对违反强制性标准而造成不良后果以至重大事故者，由法律、行政法规规定的行政主管部门依法根据情节轻重给予行政处罚，直至由司法机关追究刑事责任。

#### (2) 推荐性标准

在生产、交换、使用等方面，通过经济手段或市场调节而自愿采用的一类标准称为推荐性标准。这类标准，不具有强制性，任何单位均有权决定是否采用，违反这类标准，不构成经济或法律方面的责任。

## 10.1.4 标准的代号和编号

### 1. 国际标准 ISO 的代号和编号

国际标准 ISO 的代号和编号的格式为：ISO+标准号+[杠+分标准号]+冒号+发布年号（方括号中的内容可有可无），例如，ISO8402：1987 和 ISO9000—1：1994，是 ISO 标准

的代号和编号。

## 2. 国家标准的代号和编号

我国国家标准的代号由大写汉字拼音字母构成，强制性国家标准代号为 GB，推荐性国家标准的代号为 GB / T。

国家标准的编号由国家标准的代号、标准发布顺序号和标准发布年代号（四位数组成）。

（1）强制性国家标准：GB XXXXX — XXXX。

（2）推荐性国家标准：GB/T XXXXX — XXXX。

## 3. 行业标准的代号和编号

（1）行业标准代号：行业标准代号由汉字拼音大写字母组成，已正式公布的行业代号有：QJ（航天）、SJ（电子）、JB（机械）、JR（金融系统）等。

（2）行业标准的编号：行业标准的编号由行业标准代号、标准发布顺序及标准发布年代号（四位数）组成，表示方法如下：

- 强制性行业标准编号：XX XXXX — XXXX；
- 推荐性行业标准编号：XX/T XXXX — XXXX。

## 4. 地方标准的代号和编号

（1）地方标准的代号：地方标准代号由大写汉字拼音 DB 加上省、自治区、直辖市行政区划代码的前两位数（如北京市 11、天津市 12、上海市 31 等），再加上斜线 T 组成推荐性地方标准；不加斜线 T 为强制性地方标准，表示方法如下所述。

- 强制性地方标准：DBXX；
- 推荐性地方标准：DBXX / T。

（2）地方标准的编号：地方标准的编号由地方标准代号、地方标准发布顺序号、标准发布年代号（四位数）三部分组成，表示方法如下所述。

- 强制性地方标准：DBXX XXX — XXXX；
- 推荐性地方标准：DBXX / T XXX — XXXX。

## 5. 企业标准的代号和编号

（1）企业标准的代号：企业标准的代号由汉字大写拼音字母 Q 加斜线再加企业代号组成，企业代号可用大写拼音字母或阿拉数字或两者兼用所组成。没有强制性企业标准和推荐企业标准之分。

（2）企业标准的编号：企业标准的编号由企业标准代号，标准发布顺序号和标准发布年代号（四位数）组成，表示方法：Q / XXX XXXX — XXXX。

### 10.1.5 国际标准和国外先进标准

#### 1. 国际标准

国际标准是指国际标准化组织（ISO）、国际电工委员会（IEC）所制定的标准，以

及 ISO 出版的《国际标准题内关键词索引 (KWIC Index)》中收录的其他国际组织制定的标准。ISO 推荐列入 KWIC 索引的 27 个国际组织。一些未列入 *KWIC Index* 的国际组织所制定的某些标准也被国际公认。

## 2. 国外先进标准

国外先进标准是指国际上有权威的区域性标准；世界上经济发达国家的国家标准和通行的团体标准；包括知名企业标准在内的其他国际上公认先进的标准。主要有：

(1) 国际上有权威的区域性标准，如欧洲标准化委员会 (CEN)、欧洲电工标准化委员会 (CENELEC)、欧洲广播联盟 (EBU)、亚洲大洋洲开放系统互联研讨会 (AOW)、亚洲电子数据交换理事会 (ASEB) 等制定的标准。

(2) 世界经济技术发达国家的国家标准，如美国国家标准 (ANSI)、德国国家标准 (DIN)、英国国家标准 (BS)、日本国工业标准 (JIS)、瑞典国家标准 (SIS)、法国国家标准 (NF)、瑞士国家标准 (SNV)、意大利国家标准 (UNI)、俄罗斯国家标准 (TOCTP) 等。

(3) 国际公认的行业性团体标准，如美国材料与实验协会标准 (ASTM)、美国石油学会标准 (API)、美国军用标准 (MIL)、美国电气制造商协会标准 (NEMA)、美国电影电视工程师协会标准 (SMPTE)、美国机械工程师协会标准 (ASME)、英国石油学会 (IP) 等。

(4) 国际公认的先进企业标准，如美国 IBM 公司、美国 HP 公司、芬兰诺基亚公司、瑞士钟表公司等企业标准等。

## 3. 采用国际标准和国外先进标准

采用是指采纳和应用。采用国际标准和国外先进标准是把国际标准和国外先进标准或其内容，通过分析研究，不同程度地订入（编入）我国标准，并贯彻执行。将国际标准和国外先进标准订入国家标准的主要方法有：(1) 认可法；(2) 封面法；(3) 完全重印法；(4) 翻译法；(5) 重新制定法；(6) 包括与引用法。

## 4. 采用程度的概念

采用国际标准或国外先进标准的程度，分为等同采用、等效采用和非等效采用。

(1) 等同采用：指国家标准等同于国际标准，仅有或没有编辑性修改。编辑性修改是指不改变标准技术的内容的修改。

(2) 等效采用：指国家标准等效于国际标准，技术内容上只有很小差异。编辑上不完全相同，编写方法不完全相对应。

(3) 非等效采用：指国家标准不等效于国际标准，在技术上有重大技术差异。即国家标准中有国际标准不能接受的条款，或者在国际标准中有国家标准不能接受的条款。

采用国际标准或国外先进标准，按国家标准 GB161 的规定编写。采用程度符号用缩写字母表示，等同采用 idt 或 IDT 表示，等效采用 eqv 或 EQV，非等效采用 neq 或 NEQ

表示。

- (1) 等同采用: GB XXXX—XXXX (idt ISO XXXX—XXXX)。
- (2) 等效采用: GB XXXX—XXXX (eqv ISO XXXX—XXXX)。
- (3) 非等效采用: GB XXXX—XXXX (neq ISO XXXX—XXXX)。

#### 5. 采用国际标准和国外先进标准的原则

(1) 根据我国国民经济发展的需要确定一定时期采用国际标准和国外先进标准的方向、任务。

(2) 很多国际标准是国际上取得多年实际经验后被公认的,基本上采取“先拿来用,然后实践验证,再补充修改”的模式。

(3) 促进产品质量水平的提高是当前采用国际标准和国外先进标准的一项重要原则。

(4) 要紧密结合我国实际情况、自然资源和自然条件,需符合国家的有关法令、法规和政策,做到技术先进、经济合理、安全可靠、方便使用、促进生产力发展。

(5) 对于国际标准中的基础标准、方法标准、原材料标准和通用零部件标准,需要先行采用。

(6) 在技术引进和设备进口中采用国际标准,应符合《技术引进和设备进口标准化审查管理办法(试行)》中的规定。

(7) 当国际标准不能满足要求,或尚无国际标准时,应参照上述原则,积极采用国外先进标准。

### 10.1.6 信息技术标准化

信息技术标准化是围绕信息技术开发、信息产品的研制和信息系统建设、运行与管理而开展的一系列标准化工作。其中主要包括信息技术术语、信息表示、汉字信息处理技术、媒体、软件工程、数据库、网络通信、电子数据交换、电子卡、管理信息系统、计算机辅助技术等方面标准化。

#### 1. 信息编码标准化

编码是一种信息表现形式。它对事物或概念的描述一般比自然语言要直接、简洁、准确和有力。要保证信息编码的一致性,就要对编码对象的确定、对象特性的选择、编码方法和代码设计进行标准化。

编码是一种信息交换的技术手段。为了统一编码系统,人们借助了标准化这个工具,制定了各种标准代码,如 ASCII 码(美国信息交换标准代码)。

#### 2. 条码标准化

条码是一种特殊的代码。国家标准《GB12950—91 条码系统通用术语 条码符号术语》中定义,条码是“一组规则排列的条、空及其对应字符组成的标记,用以表示一定的信息”。目前国际上广泛使用的条码是国际物品编码协会的标准化条码 EAN。我国国



家标准 GB904—91 通用商品条码的结构与 EAN 条码结构相同,由 13 位数字码以及对应的条码组成,前缀码(3 位)、制造厂商代码(4 位)、商品代码(5 位)、检验码(1 位),3 位前缀码是标识国家或地区的代码。我国的国家代码为“690”。

### 3. 汉字编码标准化

汉字编码是对每一个汉字按一定的规律用若干个字母、数字、符号表示出来。我国在汉字编码标准化方面取得的突出成就就是信息交换用汉字编码字符集国家标准的制定。该字符集共有 6 集。其中,GB2312—80 信息交换用汉字编码字符集是基本集,收入常用基本汉字和字符 7445 个。GB7589—87 和 GB7590—87 分别是第二辅助集和第四辅助集,各收入现代规范汉字 7426 个。GB/T12345—90 是辅助集,它与第三辅助集和第五辅助集分别是与基本集、第二辅助集和第四辅助集相对应的繁体字的汉字字符集。除汉字编码标准化外,汉字信息处理标准化的内容还包括:汉字键盘输入的标准化;汉字文字识别输入和语音识别输入的标准化;汉字输出字体和质量的标准化;汉字属性和汉语词语的标准化等。

### 4. 软件工程标准化

我国 1983 年 5 月成立“计算机与信息处理标准化技术委员会”,下设十三个分技术委员会,其中程序设计语言分技术委员会和软件工程技术委员会与软件相关。已得到国家批准的软件工程国家标准有:

#### (1) 基础标准

- ① 信息处理—程序构造及其表示法的约定 GB/T 13502—92;
- ② 信息处理系统—计算机系统配置图符号及其约定 GB/T 14085—93;
- ③ 软件工程专业术语标准 GB/T 11457—89;
- ④ 软件工程标准分类法 GB/T 15538—95。

#### (2) 开发标准

- ⑤ 软件开发规范 GB 8566—88 ;
- ⑥ 计算机软件单元测试 GB/T 15532—95;
- ⑦ 软件维护指南 GB/T 14079—93。

#### (3) 文档标准

- ⑧ 计算机软件产品开发文件编制指南 GB 8567—88;
- ⑨ 计算机软件需求说明编制指南 GB/T 9385—88;
- ⑩ 计算机软件测试文件编制指南 GB/T 9386—88。

#### (4) 管理标准

- ⑪ 计算机软件配置管理计划规范 GB/T 12505—90;
- ⑫ 计算机软件质量保证计划规范 GB/T 12504—90;
- ⑬ 计算机软件可靠性和可维护性管理 GB/T 14394—93;
- ⑭ 信息技术、软件产品评价、质量特性及其使用指南 GB/T 16260—96。

## 10.1.7 标准化组织

### 1. 国际标准化组织

ISO 和 IEC 是世界上两个最大、最具有权威的国际标准化组织。由 ISO 确认并公布的国际标准化组织还有国际计量局 (BIPM)、联合国教科文组织 (UNESCO)、世界卫生组织 (WHO)、世界知识产权组织 (WIPO)、国际信息与文献联合会 (FID)、国际法制计量组织 (OIML) 等 27 个国际组织。

#### (1) 国际标准化组织 ISO (International Organization for Standardization)

国际标准化组织 ISO 是世界上最大的非政府性的, 由各国标准化团体 (ISO 成员团体) 组成的世界性联合专门机构。它成立于 1947 年 2 月。

ISO 的成员团体分正式成员和通讯成员。1947 年 ISO 成立时只有 25 个成员团体, 经 50 年的发展, 现有团体 (国家标准化机构) 135 个, 其中成员团体 (正式成员) 90 个, 通讯成员 35 个、注册成员 9 个。

成员全体大会是 ISO 的最高权力机构。理事会是 ISO 常务机构, 由正、副主席、司库和 18 个理事国代表组成, 每年召开一次会议, 理事会成员任期三年, 每年改选 1/3 的成员。理事会下设若干专门委员会。ISO 按专业性质设立技术委员会, 各技术委员会根据工作需要可设立若干分委员会 (SC), SC 下面可设立若干工作组 (WG)。ISO 现有技术组织 2871 个, 其中技术委员会 (TC) 191 个、分技术委员会 (SC) 572 个、工作组 2063 个, 临时专题小组 45 个。

#### (2) 国际电工委员会 IEC (International Electrotechnical Commission)

国际电工委员会 IEC 成立于 1906 年, 是世界上最早的非政府性国际电工标准化机构, 是联合国经社理事会 (ECOSOC) 的甲级咨询组织。1947 年 ISO 成立后, IEC 曾作为一个电工部并入 ISO。1976 年 IEC 从 ISO 中分离出来, IEC 负责有关电气工程及电子领域国际标准化工作, 其他领域则由 ISO 负责。

IEC 的工作领域包括电工领域各个方面, 如电力、电子、电讯和原子能方面的电工技术等。理事会是 IEC 的最高权利机构, 会址设在日内瓦。IEC 理事会下设执行委员会和合格评定局。执行委员会负责管理技术委员会 (TC) 和技术咨询委员会。

### 2. 区域标准化组织

区域标准化组织是指同处一个地区的某些国家组成的标准化组织。其主要职能是制定、发布和协调该地区的标准。主要有:

(1) 欧洲标准化委员会 (CEN) 成立于 1961 年, 主要任务是协调各成员国的标准, 制定必要的欧洲标准 (EN), 实行区域认证制度。

(2) 欧洲电工标准化委员会 (CEN EL EC) 成立于 1972 年, 主要是协调各成员国电器和电子领域的标准, 以及电子元器件质量认证, 制定部分欧洲标准 (EN)。

(3) 亚洲标准咨询委员会 (ASAC) 成立于 1967 年, 主要是 ISO、IEC 标准的基础

上,协调各成员国标准化活动,制定区域性标准。

(4) 国际电信联盟 ITU (International Telecommunication Union): ITU 于 1865 年 5 月在巴黎成立,ITU 的目的和任务是维持和发展国际合作,以改进和合理利用电信,促进技术设施的发展及有效应用,以提高电信业务的效率。

### 3. 行业标准化组织

行业标准化组织是指制定和公布适应于某个业务领域标准的专业标准化团体,以及在其业务领域开展标准化工作的行业机构、学术团体或国防机构。

(1) 美国电气电子工程师学会 IEEE (Institute of Electrical and Electronics Engineers), 是美国规模最大的专业学会。近年该学会专门成立了软件标准分技术委员会 (SESS)。IEEE 通过的标准常常要报请 ANSI 审批,使具有国家标准的性质。因此,IEEE 公布的标准常冠有 ANSI 字头。例如,ANSI / IEEE Str 828—1983 软件配置管理计划标准。

(2) 美国国防部批准、颁布适用于美国军队内部使用的标准代号为 DOD (采用公制计量单位的以 DOD 表示) 和 MIL。

(3) 我国国防科学技术工业委员会批准、颁布适合于国防部门和军队使用的标准,代号为 GJB。例如,1988 年发布实施的 GJB473—88 军用软件开发规范。

### 4. 国家标准化组织

国家标准化组织是指在国家范围内建立的标准化机构,以及政府确认(或承认)的标准化团体,或者接受政府标准化管理机构指导并具有权威性的民间标准化团体。这些组织主要有:

- (1) 美国国家标准学会 ANSI (American National Standards Institute)。
- (2) 英国标准学会 BSI (British Standards Institution)。
- (3) 德国标准化学会 DIN: (Deutsches Institution fur Normung)。
- (4) 法国标准化协会 AFNOR (Association Francaise de Normalisation)。

## 10.1.8 ISO9000 标准简介

### 1. ISO9000 标准

ISO9000 标准是一系列标准的统称。ISO9000 系列标准由 ISO/TC176 制定。TC176 是 ISO 的第 176 个技术委员会(质量管理和质量保证技术委员会),专门负责制定质量管理和质量保证技术的标准。2000 年 12 月 15 日,ISO9000:2000 系列标准正式发布实施。

ISO9000 系列标准的质量管理模式为企业注入新的活力和生机,给质量管理体系提供评价基础,为企业进行世界贸易带来质量可信度。

### 2. ISO9000:2000 系列标准文件结构

ISO9000:2000 族标准现有 13 项标准,由四个核心标准,一个支持标准,六个技术报告,三个小册子和一个技术规范构成,四个核心标准是: ISO9000:2000 《基本原理和术语》、ISO9001:2000 《质量管理体系—要求》、ISO9004:2000 《质量管理体系—业绩

改进指南》、ISO19011:2000《质量和环境管理审核指南》。

ISO9000族质量管理体系在ISO9000:2000和ISO9004:2000标准中提及的八项质量管理原则是该标准中一个非常重要的内容,是整个ISO9000族质量管理体系标准的精髓和纲领。八项质量管理原则是:1.以顾客为中心;2.领导作用;3.全员参与;4.过程方法;5.管理的系统方法;6.持续改进;7.基于事实的决策方法;8.互利的供方关系。

### 10.1.9 ISO/IEC 15504 过程评估标准简介

ISO/IEC15504 由 ISO/IEC JTC1/SC7/WG10 与其项目组 SPICE (Software Process Improvement and Capability Etermination, 软件过程改进和能力确定) 和国际项目管理机构共同完成,并收集整理了来自20多个国家的工业、政府以及大学专家的意见和建议,同时得到世界各地软件工程师的帮助,包括与美国SEI、加拿大贝尔合作。

ISO/IEC 15504 提供了一个软件过程评估的框架。它可以被任何软件企业用于软件的设计、管理、监督、控制以及提高获得、供应、开发、操作、升级和支持的能力。

## 10.2 例题分析

【例 10-1】 按制定标准的不同层次和适应范围,标准可分为国际标准、国家标准、行业标准和企业标准等,\_\_\_\_制定的标准是国际标准。

- A. GJB      B. IEEE      C. ANSI      D. ISO

正确答案: D

分析:

ISO 是国际标准化组织,它制定的标准是国际标准。ANSI 是美国国家标准学会,只能制定美国国家标准。IEEE 是美国电气电子工程师学会,IEEE 通过的标准常常要报请 ANSI 审批,使具有国家标准的性质。GJB 是我国国防科学技术工业委员会批准、颁布适合于国防部门和军队使用的标准,属于行业标准化组织。

【例 10-2】 我国国家标准分为强制性国家标准和推荐性国家标准,强制性国家标准的代号为\_\_\_\_。

- A. ZB      B. GB      C. GB/T      D. QB

正确答案: B

分析:

我国国家标准的代号由大写汉字拼音字母构成,强制性国家标准代号为 GB,推荐性国家标准的代号为 GB/T。

【例 10-3】 \_\_\_\_是关于质量管理体系的一系列标准,有助于企业交付符合用户质量要求的产品。

- A. ISO9000      B. CMM      C. ISO1400      D. SW—CMM



正确答案: A

分析:

ISO9000 族标准是国际标准化组织 (ISO) 颁布的在全世界范围内通用的关于质量管理 and 质量保证方面的系列标准, 目前已被 80 多个国家等同或等效采用, 该系列标准在全球具有广泛深刻的影响, 有人称之为 ISO 9000 现象。

ISO1400 是国际标准化组织第 207 技术委员会 (TC207) 从 1993 年开始制定的系列环境管理国际标准的总称, 它同以往各国自定的环境排放标准和产品的技术标准等不同, 是一个国际性标准, 对全世界工业、商业、政府等所有组织改善环境管理行为具有统一标准的功能。它由环境管理体系 (EMS) 环境行为评价 (EPE)、生命周期评估 (LCA)、环境管理 (EM)、产品标准中的环境因素 (EAPS) 等 7 个部分组成。

CMM 是软件开发能力的成熟度模型 (SW—CMM) 的简称, 包括五个成熟等级, 开发的能力越强, 开发组织的成熟度越高, 等级越高。

【例 10-4】标准化是一门综合性学科, 其工作内容极为广泛, 可渗透到各个领域。标准化工作的特征包括横向综合性、政策性和\_\_\_\_\_。

- A. 统一性                      B. 灵活性                      C. 先进性                      D. 安全性

正确答案: A

分析:

标准化是在经济、技术、科学及管理等社会实践中, 对重复性事物和概念通过制定、发布和实施标准, 达到统一, 获最佳秩序和社会效益的过程。标准化工作是一项复杂的系统工程, 标准为适应不同的要求从而构成一个庞大而复杂的系统, 标准化工作的特征包括横向综合性、政策性和统一性。

【例 10-5】自标准实施之日起, 至标准复审重新确认、修订或废止的时间, 称为标准的有效期, 我国在国家标准管理办法中规定, 国家标准的有效期一般为\_\_\_\_\_年。

- A. 2                      B. 5                      C. 7                      D. 10

正确答案: B

分析:

各个国家的标准有效期不同。ISO 标准每 5 年复审一次, 平均标龄为 4.92 年; 1988 年发布的《中华人民共和国标准化法实施条例》中规定, 标准实施后的复审周期一般不超过 5 年, 即我国的国家标准有效期一般为 5 年。

【例 10-6】\_\_\_\_\_是指在经济、技术、科学及管理等社会实践中, 对重复性事物和概念通过制定、发布和实施标准达到统一, 以获得最佳秩序和最大社会效益。

- A. 标准化                      B. 标准                      C. 规范                      D. 规程

正确答案: A

分析:

标准化是在经济、技术、科学及管理等社会实践中, 以改进产品、过程和服务的适

用性,防止贸易壁垒,促进技术合作,促进最大社会效益为目的,对重复性事物和概念通过制定、发布和实施标准,达到统一,获最佳秩序和社会效益的过程。

### 10.3 思考练习题及答案

#### 思考练习题

1. 我国标准分为国家标准、行业标准、地方标准和企业标准四类, \_\_\_\_是企业标准的代号。

- A. GB                      B. QJ                      C. Q                      D. DB

2. 《计算机软件产品开发文件编制指南》(GB 8567—88)是 \_\_\_\_ 标准。

- A. 强制性国家      B. 推荐性国家      C. 强制性行业      D. 推荐性行业

3. 标准化对象一般可分为两大类,一类是标准化的具体对象,即需要制定标准的具体事物;另一类是 \_\_\_\_。

- A. 标准化抽象对象      B. 标准化总体对象  
C. 标准化虚拟对象      D. 标准化面向对象

4. 标准化的目的之一是建立最佳秩序,即建立一定环境和一定条件的最合理秩序。标准化的另一目的,就是 \_\_\_\_。

- A. 提高资源的转化效率      B. 提高劳动生产率  
C. 保证公平贸易      D. 获得最佳效益

5. 按制定标准的不同层次和适应范围,标准可分为国际标准、国家标准、行业标准和企业标准等, \_\_\_\_标准是我国各级标准必须服从且不得与之相抵触。

- A. 国际                      B. 国家                      C. 行业                      D. 企业

6. IEEE 是一个 \_\_\_\_ 标准化组织。

- A. 国际                      B. 国家                      C. 行业                      D. 区域

#### 思考练习题答案

1. C

分析: 根据《中华人民共和国标准化法》的规定,我国标准分为国家标准、行业标准、地方标准和企业标准等四类。这四类标准主要是适用的范围不同。代号分别是: 国家标准(GB)、行业标准(QJ(航天)、SJ(电子)、JB(机械)、JR(金融系统)等)、地方标准(DB)和企业标准(Q)。

2. B

分析: 我国1983年5月成立“计算机与信息处理标准化技术委员会”,下设十三个分技术委员会,其中程序设计语言分技术委员会和软件工程技术委员会与软件相关。已

得到国家批准的软件工程国家标准有 14 个,分为基础标准、开发标准、文档标准和管理标准 4 大类,计算机软件产品开发文件编制指南 GB 8567—88 属于文档标准,是推荐性国家标准。

### 3. B

分析:标准化对象一般可分为两大类,一类是标准化的具体对象,即需要制定标准的具体事物;另一类是标准化总体对象,即各种具体对象的全体所构成的整体,通过它可以研究各种具体对象的共同属性、本质和普遍规律。

### 4. D

分析:标准化的目的之一是建立最佳秩序,即建立一定环境和一定条件的最合理秩序。通过标准化在社会生产组成部分之间进行协调,确立共同遵循的准则,建立稳定和最佳的生产、技术、安全、管理等秩序,使生产活动和经营管理活动井然有序,避免混乱,达到高效率。标准化的另一目的,就是获得最佳效益。一定范围的标准,是按一定范围内的技术效益和经济效果的目标制定出来的,它不仅考虑了标准在技术上的先进性,还考虑到经济上的合理性以及企业的最佳经济效益。

### 5. A

分析:国际标准是指国际标准化组织(ISO)、国际电工委员会(IEC)所制定的标准。国际标准在世界范围内统一使用,各国可以自愿采用,不强制适用。国家标准是由政府或国家级的机构制定或批准的、适用于全国范围的标准,是一个国家的标准体系的主体和基础,国内各级标准必须服从且不得与之相抵触。

### 6. C

分析:美国电气电子工程师学会 IEEE(Institute of Electrical and Electronics Engineers)是由美国电气工程师学会(AIEE)和美国无线电工程师学会(IRE)于 1963 年合并而成,是美国规模最大的专业学会。IEEE 主要制定标准内容有电气与电子设备、试验方法、原器件、符号、定义以及测试方法等。近年该学会专门成立了软件标准分技术委员会(SESS),积极开展了软件标准化活动,取得了显著成果,受到了软件界的关注。IEEE 是典型的行业标准化组织。

## 第 11 章 知识产权基础知识

### 11.1 内容提要

#### 11.1.1 知识产权的概念与特点

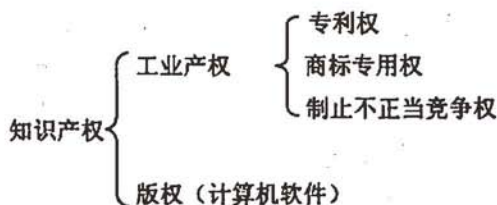
##### 1. 知识产权的概念

知识产权是指民事权利主体（公民、法人）基于自己所掌握的知识、智慧和经验，通过创造性劳动所取得的智力成果而依法产生的权利。

知识产权是一种财产。它与动产（可以移动的东西，如：手表或汽车）、不动产（土地和永久固定在土地上的东西，如：房屋、土地等）一样，是智力成果创造者所享有的一种私有财产权，或者称智慧财产权、无形的财产权。

##### 2. 知识产权的分类

按照世界知识产权组织确定的知识产权国际规则，知识产权分为工业产权和著作权两大类。具体分类情况见下表：



##### 3. 知识产权的主要特点

###### (1) 无形性

知识产权是一种知识型的产品，它是可以脱离其权利所有者而存在的无形信息。

###### (2) 专有性

知识产权是一项权利人的专有权利，具有独占性和排他性。具体表现为：未经知识产权的权利人许可（授权），任何单位或个人不得使用该知识产权，否则就构成侵权。同时，享有知识产权的权利人应当承担相应的法律责任。

###### (3) 地域性

知识产权具有严格的地域性特点，不同国家的知识产权主管机关依照本国的法律对公民、法人的智力成果授予知识产权，通常只能在本国领域内依据本国法律给予保护。



#### (4) 时间性

知识产权并不能被权利人永久占有,而是具有法定的保护期限。一旦知识产权保护期限届满,知识产权将自行终止,成为社会公众可以自由利用或者使用的智力成果。

### 11.1.2 我国保护软件知识产权的法律法规

目前,我国在短短的几十年中,已经建立了比较完备的保护软件知识产权的法律体系。

#### 1. 中华人民共和国著作权法

我国的《著作权法》(或者称版权法)与国际公约的准则一致,将计算机软件作为著作权法律保护的一类客体。

#### 2. 中华人民共和国专利法

专利权是依据专利法保护具有“新颖性、实用性和创造性(三性要求)”的技术方案和新设计。

#### 3. 中华人民共和国商标法

商标专用权是依据商标法保护实施保护。商标(Mark)与版权和专利不同,它由名称、词组、符号等单独或者任意组合。

#### 4. 中华人民共和国反不正当竞争法

我国的《反不正当竞争法》中将商业秘密定义为:“不为公众所知悉的、能为权利人带来经济利益、具有实用性并经权利人采取保密措施的技术信息和经营信息”。经营秘密和技术秘密是商业秘密的基本内容,经营秘密,即未公开的经营信息,是指与生产经营活动有关的经营方法、管理方法、产销策略、货源情报、客户名单、标底和标书内容等专有知识;技术秘密,即未公开的技术信息,是指与产品生产和制造有关的技术诀窍、生产方案、工艺流程、设计图纸、化学配方、技术情报等专有知识。

#### 5. 计算机软件保护条例

《计算机软件保护条例》是根据《著作权法》的规定、由我国国务院颁布实施保护计算机软件著作权专门性的行政法规和处理软件著作权纠纷的法律依据,它是我国保护计算机软件著作权的基本法律文件。

#### 6. 保护软件知识产权的相关法律、法规

- ① 中华人民共和国民法通则;
- ② 中华人民共和国刑法;
- ③ 中华人民共和国继承法;
- ④ 中华人民共和国合同法;
- ⑤ 最高人民法院保护知识产权的司法解释等。

上述法律是基本法,也是《计算机软件保护条例》的上位法,当《计算机软件保护条例》没有涉及的情况时,可适用其他法律的规定。

### 11.1.3 计算机软件著作权保护

#### 1. 计算机软件著作权的主体

计算机软件著作权的主体指享有软件著作权的人。根据《计算机软件保护条例》的规定,计算机软件著作权的主体包括公民、法人和其他组织。其中:

① 中国人自行开发完成的软件,采取著作权自动产生原则:中国的公民、法人和其他组织可以通过原始开发或者软件“二次开发”(“二次开发”软件要具有合法性)等智力创作活动,对自行开发完成的软件自动成为独立的著作权人;中国的公民、法人和其他组织可以通过合作开发软件等智力创作活动,对合作开发完成的软件自动成为共同著作权人。

② 外国人、无国籍人的主体资格,采取“有条件”的国民待遇原则:外国人、无国籍人的软件首先在中国境内发表,享有著作权;外国人、无国籍人的软件,依照该国与中国签定的协议或者共同参加的国际公约受到《计算机软件保护条例》保护。

③ 中国人通过订立软件委托开发合同、通过软件著作权转让合同、公民通过合法的继承文件、法人和其他组织通过合法的承受文件的途径,获得软件著作权或者著作权的财产权,取得软件著作权主体资格。

#### 2. 计算机软件著作权的客体

计算机软件著作权的客体是指《计算机软件保护条例》所确定的保护对象——计算机软件。同时,《计算机软件保护条例》也确定了著作权保护客体的法定范围、要求和条件。

##### (1) 现行的著作权法律、法规保护“商业软件”

现行的著作权法律、法规主要为“商业软件”提供法律手段,以保护软件著作权人的合法权益;而“自由软件”则是利用了现行著作权法律、法规的原则,要求自由软件遵循的是 GPL 规则。

因此,商业软件与自由软件的法律适用问题不能一概而论,采取“眉毛胡子一起抓”做法。要正确认识和区分软件著作权意义下的分类,严格界定自由软件和商业软件的法律规则和商业模式。同时,软件著作权许可合同是保护软件知识产权重要的法律规则,软件的使用者应当根据使用软件的情况,自觉遵守商业软件、共享软件、自由软件合同规则(许可证协议)。

##### (2) 计算机程序及其有关文档作为一个整体实施保护

根据我国《计算机软件保护条例》的规定,计算机程序及其有关文档作为一个整体实施保护,即:《计算机软件保护条例》保护的客体是计算机软件,而不仅仅保护计算机程序。

##### ① 计算机程序的定义

a. 计算机等具有信息处理能力的装置能够执行的;

- b. 代码化指令序列或者语句序列;
- c. 同一软件的源程序与目标程序为同一作品。

② 计算机程序有关文档的定义

a. 描述程序的内容、组成、设计、功能规格、开发情况、测试结果及使用方法的文字资料和图表;

b. 开发文档、使用文档、维护文档等。

③ 受著作权法保护的计算机软件的要求和条件

a. 受保护的软件应具有法律要求的“表达形式”。软件的源程序和相关的文档均具有表达形式,可以满足著作权法保护的要求。同时,每个软件均有版本,不同版本对应着不同表达形式、开发完成和发表时间等著作权的法律关系,要求每个软件有明确的版本号;

b. 独立开发(具有独创性)完成的软件,计算机程序及其有关文档应当同时编写(开发)完成,不要求软件具有新颖性和“完美无缺”;

c. 开发完成的软件应当固定在某种有形物体(介质)上,包括:硬盘、光盘、磁盘、集成电路等;

d. 软件应当具有可复制性,即:软件可以被重复利用。

④ 计算机软件著作权法不保护的内容

a. 开发软件所使用的思想、处理过程、操作方法、数学概念均不受著作权法的保护,它们不属于软件的表达形式;

b. 软件著作权的保护与软件所管理的内容要区别开来,如:信息、数据和资料的汇编,不属于法律、法规对软件定义的范畴,不能作为软件来保护。

3. 计算机软件著作权的权利内容

计算机软件著作权的权利内容是指软件著作权人所享有的基本权利。根据《计算机软件保护条例》规定,软件著作权人享有权利包括:

(1) 软件著作权人的基本权利

① 《条例》第八条规定软件著作权人享有的基本权利包括:发表权、署名权、修改权、复制权、发行权、出租权、信息网络传播权、翻译权等;

② 软件著作权人享有依法通过软件著作权的许可、转让,而获得报酬的权利。

(2) 软件著作权人的其他权利

① 技术措施的版权保护

对软件等作品采取加密、加锁等“反盗版技术措施”,是著作权人的专有权利,制止非法访问和通过解密、绕过(我国使用故意避开或者破坏的定义)技术措施的规避技术措施而使用软件等作品的行为。

② 权利管理电子信息的版权保护

权利管理电子信息是软件等作品的著作权人为了防止盗版,在数字作品中加入的显示



著作权人、相关权利人和版权情况等电子信息。版权管理信息是权利人专有的电子信息，不允许他人擅自破坏。

#### 4. 计算机软件著作权的归属

##### (1) 软件著作权归属的基本原则

我国《计算机软件保护条例》采取“创作主义”原则来确定软件著作权的归属。《计算机软件保护条例》明确规定在一般情况下，软件著作权属于软件开发者，除非另有规定。具体体现为：

①《计算机软件保护条例》第九条规定“软件著作权属于软件开发者，本条例另有规定的情况除外。”这是我国计算机软件著作权归属的基本原则。谁完成了计算机软件的创作开发工作，其软件的著作权就归谁享有。无论是单独开发完成的软件，还是通过合法的二次开发完成的软件，其软件的开发者便自动成为软件著作权人，而且该软件著作权人对所开发完成的软件承担相关的法定义务。

②《计算机软件保护条例》第九条第二款规定了软件中的署名推定原则。在软件上署名的自然人、法人或者其他组织为开发者。除非有相反证明能够推翻的情况下，按照软件上的署名来推定该软件的著作权人。

##### (2) 特定（协议）归属的原则

《计算机软件保护条例》确定的特定归属的原则，实际上就是以协议（合同）所确定的软件著作权归属。按照《计算机软件保护条例》的规定，有以下几种情况：

##### ① 合作开发软件的著作权归属

《计算机软件保护条例》第十条规定，合作参与软件的具体开发，并获得软件著作权。若合作开发的软件可以分割使用的，开发者对各自开发的部分享有著作权；合作开发的软件不能分割使用的，其著作权由各开发者共同享有。同时，《计算机软件保护条例》还对合作开发软件的著作权行使作了规定，即：合作开发软件的著作权人可以行使除转让权以外的其他权利。

##### ② 委托开发软件的著作权归属

《计算机软件保护条例》第十一条规定，委托开发软件要求委托人与受托人签定书面合同。软件的著作权归属按照委托人与受托人签定的书面合同约定；若委托开发书面合同或者合同中未对著作权归属作明确约定的，受托人为著作权人，并享有著作权。

##### ③ 下达任务开发软件的著作权归属

《计算机软件保护条例》第十二条规定，下达任务开发软件国家机关与接受任务的法人或者其他组织往往签定书面合同。软件的著作权归属按照签定的书面合同约定；若书面合同或者合同中未作明确约定的，接受任务的法人或者其他组织为著作权人，并享有著作权。

##### ④ 任职（兼职）开发软件的著作权归属

《计算机软件保护条例》第十三条规定，自然人在法人或其他组织任职（兼职）期



间开发完成的软件著作权归属的规定包括:

- 针对本职工作中明确指定的开发目标;
- 从事本职工作活动所预见的结果或者自然的结果;
- 主要使用了法人或者其他组织的资金、专用设备、未公开的专门信息等技术条件。

若自然人与所在的法人或者其他组织签定了知识产权归属协议,则按照协议确定软件著作权归属。否则,软件著作权属于接受任务的单位。

### (3) 软件著作权行使中产生的著作权归属

作为软件著作权人的公民的死亡、单位的变更,以及软件著作权的转让等,必然引起软件著作权归属的改变。对此,《计算机软件保护条例》对软件著作权主体的改变作了一些规定:

#### ① 公民继承的软件权利归属

《计算机软件保护条例》第十五条规定:软件著作权的继承人(自然人)可以根据《中华人民共和国继承法》的有关规定,依法享有软件著作权的除署名权以外的其他权利。

#### ② 单位变更后软件权利归属

《计算机软件保护条例》第十五条规定:作为软件著作权人的单位(法人或其他组织)发生变更(如单位的合并、破产等),合法的软件著作权承受单位享有原始著作权人所享有的各项权利。“各项权利”包括署名权等著作人身权在内全部权利。

#### ③ 权利转让后软件著作权归属

《计算机软件保护条例》第八条和二十条规定:计算机软件著作权可以全部或者部分转让,软件著作权依法发生转让后,必然带来软件著作权主体的变化,并产生新的软件著作权归属关系。

软件著作权的转让应当根据我国有关法律、法规的规定,以签订、执行书面合同的方式进行,在合同中明确软件著作权归属,并依据合同约定软件著作权归属关系。

### 5. 计算机软件著作权的行使

计算机软件著作权的行使是指软件著作权人根据《计算机软件保护条例》确定的软件著作权许可、转让,从而获取报酬的活动。

#### (1) 软件著作权的许可

《计算机软件保护条例》第八条规定,软件著作权人可以授权(许可)他人行使软件著作权,并获得报酬的权利。《计算机软件保护条例》第十八条要求许可他人行使软件著作权的,应当订立许可使用合同具体实施。未约定专有许可的,视为非专有许可;许可使用合同未明确的权利,被许可人不得行使。同时,专有许可合同的当事人,可自愿办理合同登记手续。

软件著作权许可活动的发生,不改变软件著作权的归属关系。

#### (2) 软件著作权的转让

《计算机软件保护条例》第八条和十八条规定，软件著作权人可以全部或者部分转让其享有的软件著作权，并有权获得报酬。《计算机软件保护条例》第二十条要求转让软件著作权的，应当订立书面合同具体实施。软件著作权转让合同的当事人，可以自愿办理转让合同登记手续。

软件著作权的转让活动的发生，将改变软件著作权的归属关系，建议受让人重新进行软件著作权登记。

#### 6. 计算机软件著作权的限制

软件著作权自软件开发完成之日起产生，软件著作权人可以依法处置其享有的软件著作权，并获得相应的报酬。然而，软件著作权行使不是无止境的，为了平衡软件著作权人与公众的利益，《计算机软件保护条例》规定了对软件著作权人行使权利的限制。包括：

##### (1) 软件著作权保护期限的限制

① 自然人享有著作权的软件，为终生（有生之年）加死亡后 50 年；

② 法人或者其他组织享有著作权的软件，为软件首次发表后 50 年。若开发完成之日 50 年内未发表的，不再保护。

③ 法人或者其他组织变更或者终止没有承受其权利义务法人或者其他组织的，由国家享有该软件的著作权。

##### (2) 软件著作权行使的限制

软件合法所有者无须软件著作权人的许可，具有合法复制品所有人有装入、备份、修改软件的法定权利。

##### (3) 公众享有的法定特许权

为了学习和研究软件内含的设计思想和原理，通过安装、显示、传输或者存储软件的方式使用软件的，可以不经软件著作权人许可，不向其支付报酬。

根据《著作权法》和《计算机软件保护条例》的规定，计算机软件著作权的权利自软件开发完成之日起产生，保护期为 50 年。保护期满，除开发者身份权以外，其他权利终止。一旦计算机软件著作权超出保护期，软件就进入公有领域。计算机软件著作权人的单位终止和计算机软件著作权人的公民死亡均无合法继承人时，除开发者身份权以外，该软件的其他权利进入公有领域。软件进入公有领域后成为社会公共财富，公众可无偿使用。

#### 7. 计算机软件著作权的侵权责任

计算机软件著作权是权利人的一种专有的财产权。当侵权人侵害他人的软件著作权财产权，并造成软件著作权人财产上的损失时，法律和法规将要求侵权人承担民事责任和行政责任。侵权后果严重的，法律要求侵权人承担刑事责任。

##### (1) 软件著作权的侵权行为

###### ① 一般侵权行为

《计算机软件保护条例》第二十三条规定，以下侵害他人的软件著作权的侵权行为，侵权人通常需要承担民事责任。包括：

- a. 未经软件著作权人许可发表或者登记其软件的；
- b. 将他人软件当作自己的软件发表或者登记的；
- c. 未经合作者许可，将与他人合作开发的软件当作自己单独完成的作品发表或者登记的；
- d. 在他人软件上署名或者涂改他人软件上的署名的；
- e. 未经软件著作权人许可，修改、翻译其软件的；
- f. 其他侵犯软件著作权的行为。

## ② 比较严重侵权行为

《计算机软件保护条例》第二十四条规定，以下侵害他人的软件著作权的侵权行为，侵权人不仅需要承担民事责任，还需要承担行政责任和刑事责任。包括：

- a. 复制或者部分复制著作权人软件的；
- b. 向公众发行、出租、通过信息网络传播著作权人的软件的；
- c. 故意避开或者破坏著作权人为保护其软件而采取得技术措施的；
- d. 故意删除或者改变软件权利管理电子信息的；
- e. 许可他人行使或者转让著作权人的软件著作权的。

## (2) 软件著作权的侵权责任

### ① 民事责任

侵犯计算机软件著作权的民事责任是指公民、法人或其他组织因侵犯著作权发生的后果依法应承担的法律责任。其民事责任的程度往往由人民法院根据侵权行为的情节判定。民事责任包括：承担停止侵害、消除影响、公开赔礼道歉、赔偿损失等。

在确定侵犯软件著作权的赔偿数额问题上，我国《计算机软件保护条例》第二十五条规定中增加了赔偿数额的计算方法，以及人民法院的执法权限和力度：

- a. 侵犯计算机著作权的民事责任的损失赔偿，侵权人应当按照权利人的实际损失给予赔偿；实际损失难以计算的，可以按照侵权人的违法所得给予赔偿。
- b. 赔偿数额还应当包括权利人为制止侵权行为所支付的合理开支。
- c. 权利人的实际损失或者侵权人的违法所得不能确定的，由人民法院根据侵权行为的情节，判决给予五十万元以下的赔偿。

### ② 行政责任

侵犯计算机软件著作权的行政责任是指公民、法人或其他组织因侵犯著作权发生的后果依法应承担的法律责任。我国《计算机软件保护条例》第二十四条规定了相应的行政责任，包括：

- a. 著作权行政管理部门应当责令停止违法行为，没收违法所得，没收、销毁侵权复制品。



b. 著作权行政管理部门可对侵权者处以每件一百元或者货值金额二至五倍的罚款。

### ③ 刑事责任

侵犯计算机软件著作权的刑事责任是指公民、法人或其他组织因侵犯著作权发生的后果触犯刑律的,侵权者应当承担刑事责任。我国《刑法》第二百一十七条、二百一十八条和二百二十条的规定,构成侵犯著作权罪、销售侵权复制品罪的,由司法机关追究刑事责任。

侵犯著作权罪包括:侵权数额较大的处三年以下有期徒刑;侵权数额巨大的情节特别严重处三年以上七年以下有期徒刑,并处罚金。

侵犯著作权罪程度认定标准,在《最高法院司法解释》进行了专门的规定。包括:

- a. 侵权数额较大:个人获利数 2 万;单位所得数 10 万;
- b. 侵权数额巨大:个人获利数 10 万;单位所得数 50 万;
- c. 情节特别严重:二次追究刑事责任;个人获利数 100 万;单位所得数 500 万。

### ④ 软件使用行为的法定免责规定

a. 合法持有软件复制品的单位、公民,在不经软件著作权人同意的情况下,可以根据自己使用的需要将软件装入计算机,为了存档也可以制作备份复制品,为了把软件用于实际的计算机环境或者改进其功能时也可以进行必要的修改,但是备份制品和修改后的文本不能以任何方式提供给他人(《条例》第十七条);

b. 为了学习和研究软件内含的设计思想和原理,通过安装、显示、传输或者存储软件的方式使用软件的,可以不经软件著作权人许可,不向其支付报酬(《条例》第十七条);

c. 软件复制品的出版者、制作者能够证明合法授权,发行者、出租者能够证明合法来源的,免于侵权责任(《条例》第二十八条);

d. 软件开发者能够证明可供选择的表达形式有限与现有软件相似的,不构成侵权(《条例》第二十九条);

e. 软件侵权复制品持有人在举证不知使用侵权软件(不知情)时,不承担赔偿责任,只承担停止使用、销毁等“有限”的责任(《条例》第三十条)。

## 11.1.4 计算机软件商业秘密法律保护

### (1) 计算机软件中商业秘密的法律要件

- ① 商业秘密不为公众所知悉,即该秘密具有“新颖性”;
- ② 商业秘密能够为权利人带来经济效益,即该秘密具有“价值性”;
- ③ 该项商业秘密已为其权利人采取了保密措施加以保守秘密。

商业秘密必须具备以上三个要件,缺一不可,并据此受到法律的保护。

### (2) 软件商业秘密侵权行为

根据我国《反不正当竞争法》第十条的规定,侵犯计算机软件商业秘密行为有以下



四种具体的表现形式:

① 盗窃、利诱、胁迫或其他不正当手段获取权利人的计算机软件商业秘密。

盗窃商业秘密,包括单位内部人员盗窃、外部人员盗窃、内外勾结盗窃等手段;以利诱手段获取商业秘密,通常指行为人向掌握商业秘密的人员提供财物或其他优惠条件,诱使其向行为人提供商业秘密;以胁迫手段获取商业秘密,是指行为人采取威胁、强迫手段,使他人受强制的情况下提供商业秘密;以其他不正当手段获取商业秘密。

② 披露、使用或允许他人使用不正当手段获取的计算机软件商业秘密。披露是指将权利人的商业秘密向第三人透露或向不特定的其他人公开,使其失去秘密价值;使用或允许他人使用是指非法使用他人商业秘密的具体情形。如果以非法手段获取商业秘密的行为人,将该秘密再行披露或使用,即构成双重的侵权;倘若第三人从侵权人那里获悉了商业秘密而将秘密披露或使用,同样构成侵权。

③ 违反约定或违反权利人有关保守商业秘密的要求,披露、使用或允许他人使用其所掌握的计算机软件商业秘密。合法掌握计算机软件商业秘密的人,可能是与权利人具有合同关系的对方当事人,也可能是权利人的单位工作人员或其他知情人,他们违反合同约定或单位规定的保密义务,将其所掌握的商业秘密擅自公开,或自己使用,或许可他人使用,即构成侵犯商业秘密。

④ 第三人在明知或应知前述违法行为的情况下,仍然从侵权人那里获取、使用或披露他人的计算机软件商业秘密。这是一种间接的侵权行为。

### (3) 软件商业秘密的侵权责任

根据我国《反不正当竞争法》和《刑法》的规定,计算机软件商业秘密的侵权者将承担行政责任、民事责任以及侵权行为对权利人造成重大损害的,侵权者应当承担刑事责任:

#### ① 民事责任

计算机软件商业秘密的侵权者的侵权行为对权利人的经营造成经济上的损失时,可以向人民法院提起诉讼,侵权者应当承担经济损害赔偿的民事责任。我国《反不正当竞争法》第二十条规定了侵犯商业秘密的民事责任,即经营者违反该法规定,给被侵害的经营者造成损害的,应当承担损害赔偿责任。

当计算机软件权利人的经营损失难以计算时,其赔偿数额为侵权者在侵权期间因侵权所获得的利润,侵权者同时承担被侵害者因调查该侵权者侵害其合法权益的不正当竞争行为所支付的合理费用。

#### ② 行政责任

我国《反不正当竞争法》第二十五条规定了侵犯商业秘密的行为相应的行政责任。当构成侵犯计算机软件商业秘密时,监督检查部门应当责令停止违法行为,而后根据侵权的情节依法对侵权者处以一万元以上二十万以下数额的罚款。

#### ③ 刑事责任

我国《刑法》第二百一十九条规定了侵犯商业秘密罪，包括：

- a. 侵犯商业秘密，给商业秘密权利人造成重大损失的，处三年以下有期徒刑或拘役，并处或者单处罚金；
- b. 侵犯商业秘密，给商业秘密权利人造成特别严重后果的，处三年以上七年以下有期徒刑，并处罚金；
- c. 单位侵犯商业秘密，对单位判处罚金，并对直接负责的主管人员和其他直接责任人，给予处罚。

#### (4) 商业秘密的丧失

- ① 权利人未对商业秘密采取有效的保护措施，发生了该商业秘密的泄露；
- ② 商业秘密已被他人通过独立的研究和观察所发现并掌握；
- ③ 商业秘密已发生侵害，但权利人并未对该商业秘密的侵害者采取法律行动，以排除侵害。

一项商业秘密受到法律保护的依据，是必须具备上述构成商业秘密的三个条件，当缺少上述三个条件之一都会造成商业秘密丧失保护。

## 11.2 例题分析

**【例 11-1】** 我国的某软件公司独立开发软件并进入市场销售。一段时间后，在美国发现了销售该软件侵权产品。该软件公司的软件著作权（1）在美国得到保护？该软件公司为了保护软件著作权不受侵害，维护自身的合法权益，可以依据（2）实施法律诉讼？

- (1) A. 能够                  B. 不能够                  C. 不清楚
- (2) A. 国际公约              B. 中国法律              C. 美国法律

正确答案：(1) A              (2) C

分析：

(1) 由于中国和美国共同参加了国际公约，按照国际公约所规定的各国应给予公约成员国国民待遇的原则，中国的法人和公民享有美国的法人和公民的同等待遇，中国法人的软件著作权受到侵害时，可以在美国受到法律的保护。

(2) 根据知识产权法律具有地域性特点，中国的法人和公民为保护软件著作权不受侵害，维护自身的合法权益，需要依据美国的法律提起侵权法律诉讼，而不是中国的法律，也不是国际公约。

**【例 11-2】** 随着一项软件产品（成果）的产生，直接为该软件开发带来何种知识产权？

- A. 专利权、商标专用权、著作权、商业秘密专有权
- B. 著作权、商业秘密专有权
- C. 专利权、著作权、商业秘密专有权

D. 专利权、商业秘密专有权

正确答案: B

分析:

由于软件具有作品、技术成果和产品的特征,随着一项软件产品(成果)的产生,将可能为该软件开发直接带来专利权、商标专用权、著作权、商业秘密专有权等知识产权。然而,专利权、商标专用权需要依法办理登记注册后才能获得,而著作权和商业秘密专有权是自动产生和软件开发者采取保护措施得到的,故随着一项软件产品(成果)的产生,直接为该软件开发带来的知识产权为著作权、商业秘密专有权。

【例 11-3】 软件通常具有商业秘密的法律特征,属于我国《反不正当竞争法》保护的内容。对软件商业秘密的保护包括\_\_\_\_两项基本内容。

- A. 软件的方法和方案
- B. 软件的技术秘密和经营秘密
- C. 软件的表达形式和构思
- D. 软件的品牌和信誉

正确答案: B

分析:

《中华人民共和国反不正当竞争法》中所称的商业秘密,是指不为公众所知悉、能为权利人带来经济利益、具有实用性并经权利人采取保密措施的技术信息和经营信息。据此,软件商业秘密的保护包括软件的技术秘密和经营秘密两项基本内容。

【例 11-4】 某软件公司出资、组织、并安排公司的研发人员具体开发了一套人事方面的管理软件,由该软件公司对开发完成的软件承担责任。该软件的开发者是\_\_\_\_\_。

- A. 公司的员工为软件的开发者
- B. 公司的员工为开发者,公司视为作者
- C. 软件公司是软件的开发者
- D. 软件公司和研发人员是软件的共同开发者

正确答案: C

分析:

根据《计算机软件保护条例》第三条第三项的规定:软件开发者是指实际组织开发、直接进行开发,并对开发完成的软件承担责任法人或者其他组织;或者依靠自己具有的条件独立完成软件开发,并对软件承担责任的自然人。本题中,该软件是由公司出资、组织、并安排公司的研发人员具体开发,由公司对开发完成的软件承担责任,满足软件开发者的条件。该软件公司自然成为软件的开发者。

【例 11-5】《计算机软件保护条例》所称的计算机软件是指\_\_\_\_\_。

- A. 计算机程序及其有关文档
- B. 计算机程序

- C. 计算机程序及其技术标准
- D. 计算机软件的设计思想和算法

正确答案: A

分析:

根据《计算机软件保护条例》第二条规定,本条例所称的计算机软件是指计算机程序及其相关文档。《计算机软件保护条例》第三条第一款的规定,计算机程序是指为了得到某种结果而可以由计算机等具有信息处理能力的装置执行的代码化指令序列,或者可被自动转换成代码化指令序列的符号化语句序列。计算机程序包括源程序和目标程序,同一程序的源程序文本和目标程序文本视为同一软件作品。《计算机软件保护条例》第三条第二款规定:文档是指用自然语言或者形式化语言所编写的文字资料和图表,用来描述程序的内容、组成、设计、功能规格、开发情况、测试结果及使用方法等文字资料和图表等,如程序设计说明书、流程图、用户手册等。按照《计算机软件保护条例》的规定,所称的计算机软件是指计算机程序及其有关文档。

【例 11-6】某单位在职的小章,在做好本职工作之余,受聘于其他单位并利用该单位提供的技术物质条件开发完成了一种应用软件。该软件著作权属于\_\_\_\_\_。

- A. 小章的在职单位所有
- B. 小章的兼职单位所有
- C. 小章的在职单位和兼职单位共同所有
- D. 小章个人所有

正确答案: B

分析:

根据《计算机软件保护条例》第十三条的规定:公民在单位任职期间所开发的软件,如果是执行本职工作的结果,即针对本职工作中明确指定的开发目标所开发的;或者是从事本职工作活动所预见的结果或者自然的结果,则该软件的著作权属于该单位;或者主要使用了单位的专用设备、未公开的专门信息等物资技术条件所开发并由法人或者其他组织承担责任的软件。因此,当自然人作为某单位的雇员时,其开发的软件属于执行本职工作的结果,该软件著作权应当归单位享有,不能属于该雇员个人享有。本题中小章在做好本职工作之余,受聘于其他单位并利用该单位提供的技术物质条件开发完成的一种应用软件,不是执行在职单位本职工作的结果,而是利用了受聘单位提供的技术物质条件,该软件著作权就不属在职单位享有,而是由受聘单位所有。

【例 11-7】在我国颁布实施的《计算机软件保护条例》中,对于法人或者其他组织享有著作权的软件,保护期限是 50 年,截止到软件\_\_\_\_\_第 50 年的 12 月 31 日,但软件自开发完成之日起 50 年内未发表的,《计算机软件保护条例》不再对软件保护。

- A. 法人或者其他组织成立之后
- B. 软件首次发表后



- C. 软件开发完成后
- D. 法人或者其他组织变更、终止之后

正确答案: B

分析:

根据《计算机软件保护条例》第十四条第一款规定:软件著作权自软件开发完成之日起产生。同时,《计算机软件保护条例》第十四条第三款规定:法人或者其他组织软件的著作权,保护期限是 50 年,截止到软件首次发表后第 50 年的 12 月 31 日,但软件自开发完成之日起 50 年内未发表的,本条例不再保护。这是对法人或者其他组织所享有软件的著作权的限制,需要引起重视。

【例 11-8】老王开发完成了一种控制汉字输入方法应用软件,后老王于 2000 年 3 月 10 日去世。该软件的保护期应截止到\_\_\_\_\_。

- A. 2050 年 12 月 31 日
- B. 2050 年 3 月 10 日
- C. 2000 年 3 月 10 日
- D. 2000 年 12 月 31 日

正确答案: A

分析:

根据《计算机软件保护条例》关于自然人享有著作权的软件,为终生(有生之年)加死亡后 50 年;法人或者其他组织享有著作权的软件,为软件首次发表后 50 年。若开发完成之日 50 年内未发表的,不再保护;以及法人或者其他组织变更或者终止没有承受其权利义务法人或者其他组织的,由国家享有该软件的著作权等软件著作权保护期限的限制的规定。自然人的老王于 2000 年 3 月 10 日去世之后,老王所享有著作权的软件保护期应当为 50 年(2000 年至 2050 年),即:截止到 2050 年 12 月 31 日。

【例 11-9】大学生王某,对某办公软件的技术保护措施进行解密,并将其解密后的版本制作成光盘,在中关村电子市场进行销售,破坏正常的市场秩序,损害广大消费者利益,王某的法律责任应当\_\_\_\_\_。

- A. 承担民事责任
- B. 承担民事责任及行政责任
- C. 承担民事责任及行政责任
- D. 承担民事责任及行政责任的同时,如果销售数额巨大还将承担刑事责任

正确答案: D

分析:

根据《计算机软件保护条例》第二十四条规定,未经软件著作权人或其合法受让者的许可,复制或部分复制著作权人软件的;向公众发行、出租、通过信息网络传播著作权人的软件的;故意避开或者破坏著作权人为保护其软件而采取技术措施的;故意删除

或者改变软件权利管理电子信息的；许可他人行使或者转让著作权人的软件著作权的侵权行为，软件著作权侵权人将同时承担民事责任、行政责任和刑事责任。因此，大学生王某，对某办公软件的技术保护措施进行解密，并将其解密后的版本制作成光盘，在市场进行销售，破坏正常的市场秩序，损害广大消费者利益，王某应当为自己的侵权行为承担民事责任及行政责任的同时，如果销售数额巨大还将承担刑事责任。

**【例 11-10】** 某软件公司将与他人共同开发、共同享有著作权的软件，作为自己单独开发的软件向国家软件著作权登记机构申请软件著作权登记。其他软件著作权人发现后，可以依法追究该软件公司的\_\_\_\_\_。

- A. 民事责任
- B. 行政责任
- C. 民事责任和行政责任
- D. 刑事责任

正确答案：A

分析：

这样的情况通常发生在软件的合作开发者之间，属于侵犯软件著作权的行为。根据《计算机软件保护条例》第二十三条三项的规定：未经合作者的同意将与他人合作开发的软件当作自己独立完成的作品发表或者登记的，这样的侵权行为仅承担停止侵害、消除影响、公开赔礼道歉、赔偿损失等民事责任。因此，该软件公司将为自己的行为承担代价，而不需要承担行政责任、刑事责任。

## 11.3 思考练习题及答案

### 思考练习题

1. 我国目前实施的《计算机软件保护条例》是由\_\_\_\_\_修订并公布的，自\_\_\_\_\_起施行。
2. 根据《计算机软件保护条例》的规定，计算机软件著作权自\_\_\_\_\_起产生。
3. 根据《计算机软件保护条例》的规定，软件著作权人可以向\_\_\_\_\_认定的软件登记机构办理登记。
4. 根据《计算机软件保护条例》的规定，计算机软件著作权人或者软件著作权有关权利人在制止软件著作权的侵权行为时，为了防止软件的侵权证据可能灭失、侵权证据难以取得，可以在起诉者前向\_\_\_\_\_申请保全证据。
5. 根据《计算机软件保护条例》的规定，软件合法复制品的所有人享有\_\_\_\_\_的权利。
6. 根据《计算机软件保护条例》的规定，为了\_\_\_\_\_，通过安装、显示、传输或者

储存软件方式使用软件的，可以不经软件著作权人许可，不向其支付报酬。

7. 王某是一名程序员，他未经许可对某软件公司的办公软件的技术保护措施进行了解密，并将其解密后的版本制作成光盘，在某电子市场进行销售。王某行为的法律责任应当是\_\_\_\_\_。

8. 软件通常具有商业秘密的法律特征，属于是我国《反不正当竞争法》保护的内容。对软件商业秘密的保护包括\_\_\_\_\_两项基本内容。

### 思考练习题答案

1. 国务院、2002年1月1日
2. 软件开发完成之日
3. 国务院著作权行政管理部门
4. 人民法院
5. 将软件安装到计算机等信息处理装置内、为防止软件复制品意外损坏而制作备份复制品、为使用而修改软件复制品
6. 学习和研究某办公软件内含的设计思想和原理
7. 承担民事责任及行政责任的同时，如果销售数额巨大还将承担刑事责任
8. 软件的技术秘密和经营秘密

## 第 12 章 C/C++语言程序设计

### 12.1 内容提要

C 语言是现在最流行的高级编程语言，C++是 C 语言功能的扩充和改良。C/C++语言功能强大，兼备汇编语言的功能，所以常用来编写设备驱动程序。用 C/C++语言所书写的程序代码紧凑高效。由于篇幅所限，这里不可能像通常的 C/C++语言教材那样详细讲解 C/C++的每一个功能，而且本书的读者应该都是学过 C/C++语言的，只是要全面复习。所以本章将内容浓缩，不再面面俱到，而是以若干专题的形式，讲清楚 C/C++语言的核心功能，以及在实际使用中可能遇到的问题。

#### 12.1.1 C 程序的构成

在开始介绍 C 语言的内容之前，先讲一个例子，了解一个 C 语言程序的构成，程序中用注释说明了各个语句的功能。

【例 12-1】编写程序，其功能是：从程序命令行输入的文件 data1.dat 中读出整数，求平方后以每行 10 个输出在屏幕上。

程序由两个文件组成：type.h 和 example.c，type.h 俗称头文件，进行变量的说明和定义，但只能被别的 C 文件引用，而不能直接编译；example.c 是程序文件，在头部用 include 引用 type.h 文件，程序由 3 个函数组成：main、input 和 output，其中 main 被称为主函数，是任何一个 C 程序必不可少的，input、output 分别是输入函数、输出函数，由 main 函数调用。

```
type.h:
# define      N      10000    /* 宏定义，定义常量 N */
# define      f(x)    x*x      /* 宏定义，定义函数 f(x) =x2 */
example.c:
# include     <stdio.h>        /* 引用系统文件，标准输入输出，包括文件操作 */
# include     <malloc.h>       /* 引用系统文件，动态内存分配 */
# include     "type.h"         /* 引用用户文件，即前一个文件 */
int a[N], *b;                  /* 外部变量，各个函数都可以直接访问 */
int input( );                   /* 函数向前引用说明，使以下各个函数都可以调用 input */
void main(int argc, char *argv[]) /* 主函数，带命令行参数 */
{   void      output( ); /* 函数向前引用说明，使主函数可以调用 output */
```



```

int n, i;          /* 局部变量 */
if (argc != 2)     /* 判断命令行参数个数是否为 2, 即是否输入了要显示的文件名 */
{
    printf("Usage: example <filename>\n");    /* 提示正确使用方式 */
    exit(0);    /* 终止程序运行 */
}

n = input(argv[1], a); /* 调用函数从命令行参数指定的文件中读入数据到 a 数组
                        中, 并返回读入的整数个数 */

b = (int *)malloc(sizeof(int) * n); /* 按读入的整数个数给 b 分配动态内存 */
for (i=0; i<n; i++)                /* 循环, i 从 0 直到 n-1, 步长为 1 */
    b[i] = f(a[i]);                /* 循环调用宏定义函数求每个数的平方 */
output(b, n);                      /* 调用函数 output 输出 b 数组中的整数 */
}

int input(char *filename, int a[]) /* 函数头部, 参数 1 为要读入的文件名,
                                   参数 2 为存放整数的数组, 函数返回值为所读入的整数个数 */
{
    FILE *fp; /* 文件指针, 用于打开文件, FILE 类型已在 stdio.h 中说明 */
    int i=0; /* 局部变量, 初始值置为 0, 记录读入数据的个数 */
    fp = fopen(filename, "r"); /* 以文本读方式打开指定的文件 */
    if (fp == NULL)            /* 若文件指针为空, 说明指定文件无法打开 */
    {
        printf("can't open file: %s\n", filename);
        /* 提示所指定文件无法打开 */
        exit(0);    /* 终止程序运行 */
    }

    while (!feof(fp) && i < N) /* 若未读到文件尾, 并且所读入数据个数 */
        fscanf(fp, "%d", &a[i++]); /* 未超过所定义数组的最大值, 继续读入数据 */
    fclose(fp);                /* 关闭所打开的文件 */
    return(i);                 /* 返回所读入整数的个数 */
}

void output(int c[], int n) /* 函数头部, 参数 1 为要输出的整数数组, 参数 2
                             为要输出的整数个数, 此函数无返回值 */
{
    int i; /* 局部变量, 循环变量 */
    for (i=0; i<n; i++) /* 循环, i 从 0 直到 n-1, 步长为 1 */
    {
        printf("8d", c[i]); /* 按场宽 8 输出整数 */
        if (i%10 == 0) printf("\n"); /* 若 i 为 10 的倍数, 则输出一个回车换行 */
    }
}

```

此程序的执行方式是:

c:> example data1.dat <回车>

此程序中提到的各个语句和功能，下面分专题逐个介绍。

在上面的程序中可以看到，一个C程序可以由多个部分组成，有些可以因需要而取舍，有些则是必不可少的。在一个C程序中，main函数是必不可少的，在一个程序中，无论你将main函数放置在什么位置，运行此程序时都是从main处开始执行。

若将其他部分都省略，就可以得到一个最小的C程序：

```
main( ){} 
```

此程序可以编译运行，但没有任何结果。若想有输出结果，则至少需要有一个输出语句，则程序可变为：

```
main( ){ printf("OK!\n"); }
```

此程序编译运行的结果是：OK!

上面的程序是一个有输出结果的最小C程序，其他C语句和功能可以在此基础上扩充。

### 12.1.2 变量的定义

C程序中用到的任何变量名或常量名在使用前都必须定义，常量名可以用宏定义来定义，在后面专门有一节介绍宏定义。这里只介绍变量的定义和使用。

变量分为两大类：外部变量和内部变量。外部变量在函数的外面定义，本文件的任何函数甚至其他文件的任何函数都可以访问，这不太符合结构化程序的要求，而且安全性可靠性大打折扣，所以建议少用外部变量，但由于使用方便，使用还是很广泛。

内部变量在函数中和复合语句的{ }中定义和使用，格式通常是：

```
{  变量定义
    使用变量的语句组
}
```

需要注意的是，变量的定义必须在任何执行语句前。在任何执行语句出现以后，不能再出现变量定义（但C++中允许这样），但变量的初始化除外。

在任意"{"后，都可以定义新的变量，其作用范围是到与这个"{"对应的那个"}"为止。在一个{ }段内定义的变量名可以与外面的变量名相同，但在这个{ }段内新定义的变量会掩蔽同名的外部变量，到"}"为止，这个新定义的同名变量就不再存在，恢复外面的同名变量，其值不受影响。

变量的定义格式通常是：

数据类型 变量名1，变量名2，…，变量名n；

变量名间用逗号'，'，最后以分号';'结束。

### 12.1.3 数据类型

C 语言的数据类型分为：基本类型和复合类型。基本类型包括：算术类型、枚举类型和 void 类型。

#### (1) 算术类型

包括字符型、整型和实型。

##### • 字符型

字符型占 1 个字节，被称为字符型的原因是它恰好可以存储一个西文字符。可以将字符型变量当作整型使用进行运算，只是要注意这种类型的整数的取值范围比较小。

char: 取值范围为  $-128 \sim 127$  ( $-2^7 \sim 2^7 - 1$ )

unsigned char: 取值范围为  $0 \sim 255$  ( $0 \sim 2^8 - 1$ )

##### • 整型

整型包括：short（短整型）、unsigned short（无符号短整型）、int（整型）、unsigned int（无符号整型）、long（长整型）、unsigned long（无符号长整型）。在 VC++4.0 以上的编译系统上还有 INT64 类型，即 64 位的长整型，由于使用不是很广泛，这里不介绍。

short: 16 位和 32 位编译系统都占 2 个字节，取值范围都为  $-32768 \sim 32767$  ( $-2^{15} \sim 2^{15} - 1$ )。

unsigned short: 16 位和 32 位编译系统占 2 个字节，取值范围都为  $0 \sim 65535$  ( $0 \sim 2^{16} - 1$ )。

int: 16 位编译系统占 2 个字节，取值范围都为  $-32768 \sim 32767$  ( $-2^{15} \sim 2^{15} - 1$ )。

32 位编译系统占 4 个字节，取值范围为  $-2147483648 \sim 2147483647$  ( $-2^{31} \sim 2^{31} - 1$ )。

unsigned int: 16 位编译系统占 2 个字节，取值范围都为  $0 \sim 65535$  ( $0 \sim 2^{16} - 1$ )。

32 位编译系统占 4 个字节，取值范围为  $0 \sim 4294967295$  ( $0 \sim 2^{32} - 1$ )。

long: 16 位和 32 位编译系统都占 4 个字节，取值范围都为

$-2147483648 \sim 2147483647$  ( $-2^{31} \sim 2^{31} - 1$ )。

unsigned long: 16 位和 32 位编译系统都占 4 个字节，取值范围都为

$0 \sim 4294967295$  ( $0 \sim 2^{32} - 1$ )。

注意：

(a) 通常 int 在 16 位编译系统上与 short 一致，在 32 位编译系统上与 long 一致。

(b) long 和 short 都是类型简写，可以写成 long int 和 short int，而对应的无符号类型可以写成 unsigned long int 和 unsigned short int。需要注意单词顺序不能颠倒。

(c) 给 long 型变量赋值时，在 16 位编译系统上需要在整数的尾部加上 l 或 L，表示是长整数。

(d) long 型变量输出时，在 16 位编译系统上需要用 “%ld” 才能正确输出。

##### • 实型

float: 占 4 个字节，被称为单精度，取值范围为： $-10^{38} \sim 10^{38}$ ，7 位有效数字。

**double**: 占8个字节, 被称为双精度, 取值范围为:  $-10^{308} \sim 10^{308}$ , 15位有效数字。

C语言在实型计算时, 都按照 **double** 来计算, 以防止丢失精度。

**注意**: 在 VC++6.0 编译器上 (32 位), 由于任何实数都认为是 **double** 型, 所以要将一个实数赋值给一个 **float** 变量, 需要在实数的尾部加上 **f**, 声明此实数是 **float** 型, 否则编译时系统会出现警告错误, 认为将一个 **double** 数赋值给 **float** 变量会丢失精度。

## (2) 枚举类型

关键字是 **enum**, 是为便于类别判断而设置的类型, 很少使用, 由于篇幅所限, 这里不介绍。

## (3) void 类型

无类型, 用于没有返回值的函数说明, 确保没有返回值。若相应函数中出现 **return** 语句, 系统会提示错误信息。便于编译系统帮助查错。

## (4) 复合类型

包括: 指针、数组、函数、结构体、共同体。复合一词是相对于基本类型而言, 这里不一一详细介绍, 在下面各节会逐个介绍。

# 12.1.4 算术表达式

算术表达式用到的算术运算符包括: **+**、**-**、**\***、**/**、**%**。

其中 **+**、**-** 的运算优先级相同, **\***、**/**、**%** 的运算优先级相同, 后一组的运算符的优先级高于前一组, 同组的运算符按从左到右的顺序计算, 但小括号 **()** 可以改变优先级, 小括号 **()** 中的表达式优先级最高。

**注意**:

- **%** 是求余运算, 只能对整数运算, 但包括正整数和负整数, 例如  $15\%2=1$  仍为正整数,  $(-9)\%2=-1$  仍为负整数。
- 任何两个整数的算术运算的结果仍是整数, 例如:  $3/2$  的结果是 1 而不是 1.5, 这在编程时要小心。
- 表达式中仅可以用小括号 **()**, 但不能使用中括号 **[]** 和大括号 **{}**。

**【例 12-2】** 对于程序段

```
float f; int n=4;  
f=1/n; printf("%f\n", f);
```

输出结果是: 0.000000

**f** 的值是 0.0 而不是 0.25, 与 **f** 是否是 **float** 型无关, 即使用强制类型转化写成:  $f=(float)(1/n)$ ; 也是这样。正确的赋值语句是:  $f=1.0/n$ ; 或  $f=(float)1/n$ ; 运算结果总是向精度最高的类型看齐。



### 12.1.5 赋值表达式

赋值表达式通常的格式是：

<变量> 赋值运算符 <算术表达式>

其中用到的赋值运算符包括：=、+=、-=、\*=、/=、%=，赋值运算符是将算术运算符和等号组合而成的。上式等价于：

<变量> = <变量> 算术运算符 (<算术表达式>)

执行时先计算右侧的算术表达式，再与变量进行相应的计算。

### 12.1.6 ++、--和逗号运算符

(1) ++和--运算符

++i 和 i++ 都等价于 i = i+1;

--i 和 i-- 都等价于 i = i-1;

表面结果是这样，但其中存在执行时序问题，即是先加减后使用，还是先使用后加减。

注意：

- 操作符++和--是一元操作符，两个符号之间不能有空格；
- 一元操作符的执行优先级高于所有二元操作符。

(2) 逗号运算符

逗号“,”操作符是让多个语句并行执行，通常用在不能多次出现分号“;”的语句中，例如：for (i=0,j=0; i<n; i++,j++)

该语句中，i=0,j=0;和 i++,j++都是并行执行，是为了在一个循环语句中同时有两个循环变量。

逗号“,”语句的执行顺序是从左向右，最右边的（最后一个）语句值是此表达式的值，例如：n = (i=1, j=2, k=3); 最后 n=3。

注意：逗号操作符的执行优先级低于所有二元操作符。

### 12.1.7 三目运算符

三目运算符的格式通常为：a>b ? a : b

当a>b为真时，取a为此表达式值；否则取b为此表达式值。

语句：

```
c = a > b ? a : b;
```

等价于：

```
if (a > b)    c = a;
else        c = b;
```

使用三元运算符使语句更简洁明了, 例如:

```
d = a > b ? (a > c ? a : c) : (b > c ? b : c);
```

是取 a, b, c 中值最大的给 d, 比用 if 语句:

```
if (a > b)
    if (a > c)    d = a;
    else        d = c;
else if (b > c)  d = b;
else            d = c;
```

要简洁许多。

注意: 三元运算符的执行优先级低于所有二元操作符, 仅高于逗号运算符。

### 12.1.8 输入/输出

C 语言的输入/输出语句相对于别的高级程序语言来讲, 功能强大, 数量也很多。但严格来说, 输入/输出语句并不属于 C 语言编译器的核心内容, 而是属于 C 语言的库函数 `stdio.h` (标准输入输出), 所以输入/输出语句名并不是关键字。但编写任何一个程序, 必须有输入输出, 所以标准输入输出函数成了 C 语言不可分割的组成部分, 许多书也不再区分, 把它们当成 C 语言的一部分来介绍, 称之为输入/输出语句。

每个输入/输出语句名就是一个函数名, 每使用一个输入/输出语句实际上是进行一次函数调用, 这也是为什么任何一个程序的开头都要用 `#include` 语句引用 `stdio.h`, 这是为可能调用输入输出函数进行向前引用说明。

明白了输入输出语句是函数调用这一点, 也就不难理解为什么像 `scanf` 语句, 为了读入数据到一个变量中, 需要给出这个变量的地址, 因为 C 函数的参数是传值的, 不能通过参数本身传回值。

#### (1) 输出语句

常用的输出语句有: `printf`, `putchar`, `putc`, `write` 等, 这里只介绍常用的 `printf` 和 `putchar` 语句。

- `putchar` 语句

`putchar` 语句就是简单地输出一个字符到标准输出设备上, 格式是:

```
int putchar(char ch)
```

若正确输出, 则 `putchar` 的返回值是输出的字符, 否则返回 EOF (-1)。

- `printf` 语句

`printf` 语句的格式通常为: `printf(" %6d %12.6f\n", i, f);`

语句中" "中的部分为输出格式控制, 其中:

**%c:** 一个字符。

**%d:** 十进制整数, **%ld** 为长整型(16 位编译器上必须使用), **%hd** 为短整型, **%l64d** 为 64 位长整数(VC++4.0 以上输出 INT64 类型的整数)。

**%o:** 输出八进制格式整数, 但不带先导 0。例如十进制数 15 输出为 17。

**%#o** 加先导 0, 例如十进制数 15 输出为 017。

**%x, %X:** 输出十六进制格式整数, 但不带先导 0x 或 0X。例如十进制数 2622 用 **%x** 数据格式输出为 a3e, 用 **%X** 数据格式输出为 A3E。

**%#x, %#X:** 输出带先导 0x 或 0X 的十六进制数。例如十进制数 2622 用  **%#x** 数据格式输出为 0xa3e, 而用  **%#X** 数据格式输出为 0XA3E。

**%u:** 无符号十进制整数。

**%f:** 数学形式浮点数, 例如, **%12.6f** 输出为 -123.456789。

**%e, %E:** 科学形式浮点数, 例如, 实数 -145.6789 用 **%12.6e** 输出结果为 -1.456789e+02, 而用 **%12.6E** 输出结果为 -1.456789E+02。

**%s:** 一个字符串。

以上格式都是右对齐, 若想左对齐, 在 % 后加一个 'l'。另外, 还可利用 '\n' (回车)、'\r' (回行但不回车)、'\t' (制表)、'\a' (响铃) 等控制格式。

在 % 后加数字, 是为了控制输出的场宽, **printf("%6d\n", i);** 是将整数 i 按场宽 6 输出, 若 i 的数值不足 6 位, 则在其前面补空格; 若是按 **"%06d"** 输出, 若 i 的数值不足 6 位, 则在其前面补零。但当实际 i 的数值超过 6 位时, 将按实际的位数输出, 不再受此场宽的限制。

而 **printf("%12.6f\n", f);** 是为了按总场宽 12 而小数位数为 6 的格式输出一个浮点数 f; 若浮点数的小数位数不足 6 位, 则自动用零补足。总场宽 12 位包括小数点, 所以留给浮点数整数部分的场宽是 5 位 (包括符号位), 若浮点数整数部分的位数不足 5 位, 则在其前面补空格; 若是按 **"%012.6f"** 输出, 若浮点数整数部分的位数不足 5 位, 则在其前面补零。但当浮点数整数部分的实际位数超过 5 位时, 将按实际的位数输出, 不再受此场宽的限制。

此外, 还可以变场宽输出: **printf("%\*d", m, i);** 按照 m 指定的场宽输出 i, m 不输出。而 **printf("%\*.\*f", m, n, f);** 按照 m, n 指定的场宽输出浮点数 f, m, n 不输出。

注意: C 语言的输出格式完全是自由格式, 一切要由你来控制。

## (2) 输入语句

常用的输入语句有: **scanf, getchar, getc, read** 等, 但这里只介绍常用的 **scanf** 和 **getchar** 语句。

- **getchar** 语句

**getchar** 语句是从标准输入设备上输入一个字符, 格式是:

```
char getchar()
```

若出错或文件结束，getchar 的返回值是 EOF。

- scanf 语句

scanf 语句的格式为：

```
scanf("%d %d%f%s", &i, p, &f, a);
```

其中 i, f, a 的类型是：int i, j, \*p=&j; float f; char a[100];

注意：为读入数据，需要传变量地址给 scanf 才能读入，例如 &i, &f，但数组名 a 本身是地址，不需要再取其地址，指针 p 也同样本身是地址，不需要再取其地址。

若使用通配符，则输入数据时，要按其格式输入，例如：

```
scanf("a=%d, b=%d", &a, &b);
```

输入数据时不再是简单地输入两个整数，而是要输入：

```
a=12,b=34
```

### 12.1.9 选择结构 if

C 语言没有逻辑类型，是用整数 0 和 1 表示逻辑的真假，0 表示假，1 表示真。通常一个表达式的值为 0，则认为假，结果为 0；否则认为真，结果为 1。

#### (1) 关系运算符

C 关系运算符有：<（小于）、<=（小于等于）、>（大于）、>=（大于等于）、==（等于）、!=（不等于）。两个符号之间不可以有空格，结合从左向右。

对于：a=5, b=4, c=3，并判断是否 a>b>c，如果 C 语句写成 if (a>b>c) 编译时不会有语法错误，但结果不对，因为 a>b 为真，结果为 1，而 1>c 为假。正确的写法是 if (a>b && b>c)，这样结果为真。

注意：关系运算符的执行优先级低于算术运算符，但高于三目运算符。

#### (2) 逻辑运算符

C 逻辑运算符有：&&（与）、||（或）、!（非），其中!为一元操作符，优先级高于所有二元操作符，&&优先级低于关系运算符，但高于||的优先级。

需要注意 C 语言为了提高执行效率而采用的规则是：

- 在有连续几个&&的表达式中，从左向右，只要有一个关系运算结果为假，整个结果将为假，则不再执行后面的关系运算。例如若 a=5, b=4, c=3，当执行语句：b>a && c++>a 时，由于 b>a 为假，故不再执行 c++>a，所以此语句执行完 c 仍为 3。
- 在有连续几个||的表达式中，从左向右，只要有一个关系运算结果为真，整个结



果将为真, 则不再执行后面的关系运算。例如若  $a=5$ ,  $b=4$ ,  $c=3$ , 当执行语句:  $a>b \parallel c++>a$  时, 由于  $a>b$  为真, 故不再执行  $c++>a$ , 所以此语句执行完  $c$  仍为 3。

- 在 C 语言程序中, 常会看到: `if(a)` 语句,  $a$  是一个整数, 它等价于 `if(a != 0)` 语句。而对于 `while(*p)` 语句,  $p$  是一个字符串指针, 它等价于 `while(*p != NULL)`, 即判断是否到串尾, 未到串尾继续循环。同样 `if(!a)` 语句等价于 `if(a == 0)` 语句。

但当  $a$  为浮点数时, 要小心使用这种判断语句, 因为误差的原因通常不能直接判断是否为 0.0, 若直接用 `if(a==0.0)`, 即使  $a$  的值是 0.0, 也不一定会为真。而应该使用语句 `if(fabs(a) < 1e-20)` 来判断是否达到了精度要求  $|a| < 10^{-20}$ 。

### (3) if 语句

if 语句的格式通常是:

`if(条件表达式)`

    语句体 1

`else`

    语句体 2

当<条件表达式>为真(非 0)则执行<语句体 1>, 否则执行<语句体 2>, 但 `else` 部分可以不出现。当<语句体 1>为空时, 也要写分号“;”即

`if(a>b);`

`else     a=b;`

多个 if 语句嵌套使用时, `else` 与最近的 if 结合。

## 12.1.10 switch 语句

switch 语句又称为多路选择语句。通常格式是:

`switch(表达式)`

`{ case 常量表达式 1: 语句组 1;`

`case 常量表达式 2: 语句组 2;`

`...`

`case 常量表达式 N: 语句组 N;`

`default: 语句组 N+1;`

`}`

当<表达式>的值等于  $N$  个常量表达式中某一个, 则执行相应的语句组; 而当<表达式>的值不等于  $N$  个常量表达式中任何一个时, 则执行 `default` 的语句组  $N+1$ 。

注意: 前  $N$  个语句组中的最后一个语句都应该是 `break` 语句, 否则在执行完这个语句组后会继续顺序执行下面的语句组。得不到相应正确的结果。例如: 当  $i=1$ ,  $c=3$  时, 执行下面语句:

```
switch (i)
{case 1: c++;
 case 2: c+=2;
 default: c+=5;
}
```

后, c 的值为 11, 而不是期望的 4, 因为缺少 break 终止执行, c++; c+=2;和 c+=5 这 3 个语句都被顺序执行了。正确的写法是:

```
switch (i)
{case 1: c++; break;
 case 2: c+=2; break;
 default: c+=5;
}
```

### 12.1.11 标号语句和 goto 语句

虽然 C 语言不提倡使用 goto 语句, 许多人也认为用 goto 语句不符合结构化程序设计的要求。但 goto 语句对编程者来说有时使用还是比较方便的。例如在多重循环中为了 一次性终止多重循环, 仅仅利用 break 不够方便, 而利用 goto 语句非常方便快捷。

**【例 12-3】** 下面的程序段

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            { if (a[i][j][k] < 0) goto ERROR; }
ERROR: printf("data error! \n");
```

当数据出现负值时, 利用 goto 可以立即终止三重循环。

标号语句是在任意语句前面加上一个后缀是 ‘:’ 的标识符, 若实在没有语句, 也要放置一个空语句, 即 ‘:’ 后加一个分号 ‘;’。

**【例 12-4】** 下面的程序段, 当发生溢出时立即终止执行。

```
if (i>N)
{ printf("Overflow!\n");
  goto STOP123;
}
...
STOP123: ;
printf("Error!\n");
exit(0);
```

### 12.1.12 while 语句

while 语句的一般格式为:

```
while (<表达式>)
```

```
    <循环体>;
```

当<表达式>为真(非0)时,则执行一次<循环体>,再判断<表达式>是否为真,为真则继续执行<循环体>,为假(为0)则停止循环。

注意:

- while 语句的<循环体>可能一次都不执行。
- while 语句的<循环体>与<表达式>是相关的,<循环体>的执行应该使<表达式>向结果为真的方向靠近。
- 但有时 while 也故意构成死循环,但循环体中会另有出口。
- while 语句常用于事先并不知道循环次数的循环,例如控制精度的计算等。

【例 12-5】 下面的程序段

```
while (1)
    if (getc( ) == '#') break;
```

由于 while 的条件表达式为恒真,将一直循环等待键盘输入,直到输入字符'#'才会通过 break 语句终止循环。

### 12.1.13 do-while 语句

do-while 语句的一般格式为:

```
do {
```

```
    <循环体>
```

```
} while(<表达式>);
```

先执行<循环体>一次,再去判断<表达式>是否为真,是继续执行<循环体>;否则停止。所以 do-while 的循环体至少执行一次。

### 12.1.14 for 语句

for 语句的一般格式是:

```
for (<表达式 1>; <表达式 2>; <表达式 3>)
```

```
    <循环体>
```

其中: • <表达式 1>为赋初值,只在进入时执行一次。

• <表达式 2>为条件表达式,为真则执行一次循环体。

• <表达式 3>为后处理,向循环结束的方向前进一步。

执行循序是：(a) 执行<表达式 1>。

(b) 执行<表达式 2>，判断是否为假，是则循环结束，否则执行下一步。

(c) 执行<循环体>一次。

(d) 执行<表达式 3>，并转到步骤 (b)。

上面的 for 语句等价于下面的 while 循环：

<表达式 1>;

while(<表达式 2>)

{ <循环体>;

<表达式 3>;

}

一般解决知道循环的起始点、终止点及其循环次数的问题使用 for 语句，尤其是处理数组离不开 for 语句。

### 12.1.15 continue 和 break 语句

continue 语句立即结束本次循环，开始下一次循环。

break 语句是终止本层循环。

【例 12-6】有程序段

```
for (i=0; i<100; i++)
{
    for (j=0; j<100; j++)
    {
        if (a[i][j] == 0) continue;
        if (a[i][j] == -1) break;
    }
}
```

其中若 a[i][j] 为 0，continue 语句将结束本次循环，不再执行后面判断 a[i][j] 是否为 -1 的 if 语句；而若 a[i][j] 为 -1，break 语句将终止本次的内层循环变量为 j 的循环，开始 i+1 时的循环。

注意：break 语句只能终止一层循环。

### 12.1.16 字符型数据

'A', '\n', '0', '\0' 为字符常量，占一个字节。单个字符用单引号引起来。其中的 \ 为转义符，使字符表示别的意思，例如，'\n' 不再表示字符 'n'，而是表示回车换行；'\t' 不再表示字符 't'，而是表示制表符；'\0' 不再表示字符 '0'，而是表示字符串结束符，ASCII 内码为 0。\" 表示单引号字符，\\ 表示真正的 ASCII 表上的 \ 字符。\\ddd' 为八进制的 ASCII 字符。

'ab' 写法是错误的，单引号中只能是单个字符。



"abc", "A"表示字符串, 前一个占 4 个字节, 后一个占 2 个字节, 因为字符串尾部有一个无形的字符串结束符号: '\0' 字符, 字符串需要用双引号引起来。例如:

```
char a, b, c[10], d[10];
scanf("%c%c%s%s", &a, &b, c, d);
```

其中 a, b 为字符型, c, d 为字符串。

注意: (可以使用 `if(a==b)` 判断 a 和 b 两个字符是否相同。但不能直接判断两个字符串是否相同, `if(c == d)` 或 `if(c == "ABC")` 都是错误的。正确的方法是利用 `string.h` 库函数 `strcmp` 来判断两个字符串是否相同, `if(!strcmp(c, d))` 或 `if(!strcmp(c, "ABC"))` 都是正确的, 因为当两个字符串相同时, `strcmp` 的返回值为 0。

### 12.1.17 文件引用

`include` 是文件引用, 属于 C 语言的预处理部分, C 编译器会将引用的文件内容插入到出现 `include` 的位置。例如:

```
# include <stdio.h>
```

引用标准输入输出的文件说明, 说明某些符号 (例如 `FILE` 类型) 和函数向前引用。符号 `<>` 指引用的是系统文件 (不在用户目录中搜索)。而

```
# include "type.h"
```

引用用户自己定义的文件 `type.h`, 说明某些变量和函数向前引用。符号 `"` 指引用的是用户文件 (首先在用户目录中搜索, 找不到再到系统的 `INCLUDE` 目录去搜索)。

`include` 所引用的文件, 俗称头文件 (header file), 所以习惯用 `.h` 作为文件后缀, 这类头文件存放的一般是某些变量和函数向前引用说明和宏定义的常量, 供所有使用这些变量、常量以及调用这些函数的程序文件引用。这样做既方便了编程人员, 也使程序可靠性增强, 因为修改某些定义只需修改头文件即可。

`include` 可以多层嵌套引用, 即 `include` 引用的文件中还可以有 `include`, 编译系统将递归插入展开出现的每一个 `include` 文件。

**【例 12-7】** 如下程序有 `sind.h` 和 `type.h` 两个头文件和 `exam.c` 文件。

```
sind.h:
    # define N    100

type.h:
    #include "sind.h"
    # define M    N+1
```

```
extern int a;
extern float f(float x);
```

```
exam.c:
#include "type.h"
int a;
main( ) {...}
float f(float x) {...}
```

当系统进行编译时，首先进行预处理，将出现的每一个 include 文件插入到引用处，将 type.h 的文本替换掉 exam.c 文件中第 1 行#include "type.h"，而 sind.h 文件也将替换掉 type.h 中的#include "sind.h"，最终形成如下的程序：

```
exam.c:
# define N    100
# define M    N+1
extern int a;
extern float f(float x);
int a;
main( ) {...}
float f(float x) {...}
```

### 12.1.18 宏定义

define 是宏定义，用来定义常量或宏函数，是一个符号替换的概念。例如：

```
#define N    100
#define f(x) x*x
```

预编译时，首先把程序中出现 N 的地方统统替换成 100，而出现 f(x)的地方统统替换成 x\*x。在实际使用时，要注意后者的副作用。例如：

**【例 12-8】** 若有程序段

```
#define f(x) x*x
x=2; y = x / f(x);
```

结果 y 为 2，而不是我们想要的 y 为 0.5。原因是，预编译时将 f(x)替换成了 x\*x，结果是：y=x/x\*x；

按照算术运算符的计算顺序，首先执行 x/x，结果为 1，所以最后结果为 2。为避免出现这种问题，f(x)的宏定义应该改为：

```
#define f(x) (x*x)
```

即加上括号, 提高优先级, 并防止替换后出现二义性。但即使写成这样也还会出现问题。

用宏定义定义一个函数  $f(x)=x^2$ , 只有定义成下面的形式

```
# define g(x) ((x)*(x))
```

才真正在使用中不会出现问题。宏定义的这种问题必须引起足够重视。

### 12.1.19 函数

函数分库函数和用户自定义函数两大类。C 提供了大量库函数, 通过文件引用函数的引用说明文件 `stdio.h`, `math.h`, `string.h`, `stdlib.h`, `malloc.h` 等, 就可以正确使用库函数。下面着重介绍用户自定义函数。

#### (1) 用户函数的格式

一般格式是:

<函数类型> <函数名> (<参数表>)

{ <函数体>;

}

其中, <函数类型>为函数返回值的类型, 无返回值则应写 `void`; <参数表>为函数的接口参数, 可以为空, 即表示没有参数, 但函数名后面的括号 ( ) 不能省略。

C 语言只有函数, 而不再有过程的概念。是把函数和过程合二为一, 有返回值 (`return` 语句) 的为函数, 而无返回值的 (空 `return` 语句或者根本就没有 `return` 语句) 可以看成是过程。

#### 【例 12-9】

```
float f(float x) { return (x*x); }
```

就是定义函数  $f(x)=x^2$ , 其中的 `return` 语句中的 () 有否都可以。而:

#### 【例 12-10】

```
void g(float x) { printf(" %f ", x); return; }
```

就是定义一个打印过程 `g`, 输出一个浮点数, 其中的 `return` 语句有否都可以。

#### (2) 函数的向前引用说明

#### 【例 12-11】 有如下程序

```
main( ) { float f1, a=4; f1 = f(a); }  
float f(float x) { return x*x; }
```

上面的程序编译时会提示 `f1=f(x)` 中的函数类型和函数体的类型不匹配。原因是函数的使用在函数体的定义之前, 编译系统在编译语句 `f1=f(x)` 时, 由于 `f` 是一个前面没有说

明类型的标识符，故默认为 int 型，而遇到 float f(float x) 时，f 定义为 float 类型，故提示类型不一致。

解决的办法就是在使用前先说明函数类型，本例就应该在 main( ) 语句前或 main 中的变量说明部分加上以下说明：

```
float f(float x);
```

使程序变为：

```
float f(float x);  
main()  
{ float f1, a=4; f1 = f(a); }  
float f(float x)  
{ return x*x; }
```

或者变为：

```
main()  
{ float f1, a=4; float f(float x);  
  f1 = f(a);  
}  
float f(float x) { return x*x; }
```

编译时不会再有错误，会得到正确的结果。这种说明方式就是函数的向前引用说明。任何函数只要在定义前使用都存在向前引用说明的问题，尤其是某些间接递归程序必然出现这些问题。

有时即使不是所有的函数存在向前引用说明的问题，为了以防万一，会将所有函数名都进行向前引用说明，而且这种函数的向前引用说明常常存放在头文件中。

### (3) 参数的传递

C 语言函数之间的参数传递是传值，是通过栈来传递的。调用时所有参数在栈中新开辟相应类型的单元并将实参值填入，函数中对参数的任何操作都是对栈中单元的操作，调用结束，栈中开辟的相应单元都会释放，并不影响实参变量的值。

结果是形成单向传递，一个函数可以通过参数把变量值传递给被调用函数，但被调用函数不能通过参数把变量值传回调用它的函数。这是一种防止被调用函数破坏调用函数数据的“副作用”的掩蔽措施。

**【例 12-12】** 有如下程序

```
main( )  
{ int a=3, b=2, c=1;
```



```
f(a,b,c); printf("c=%d\n", c);  
}  
f(int a, int b, int c) { c=a+b; }
```

执行结果是: c=1。函数中的赋值 c=a+b 并没有影响主程序中的 c 值。

因此, 被调用函数只能通过传地址的方式把变量值传回调用它的函数。实际上不是什么传回值的概念, 因为将变量的地址传给了被调用函数, 被调用函数用指针方式使用这些参数, 对参数的任何操作, 不再是对栈中的单元进行操作, 而是对实际实参变量的操作。例如:

```
scanf(" %d%f ", &a, &f);
```

就是把 a 和 f 的地址传给 scanf 函数, 而读入 (传回) 数值。

C++对 C 语言函数参数只能单向传值的问题进行了改进, 使之也能双向传值, 但在函数定义时要事先声明。

### 12.1.20 数组

C 语言数组分为一维、二维和多维数组。

#### (1) 数组的定义

int a[10]; a 数组下标是从 0~9, 数组元素为 a[0]~a[9], 即如果定义一个数组长度为 N, 则其下标总是从 0~N-1, C 语言的数组下标总是从 0 开始。

同样地, int b[5][5]; 数组元素为 b[0][0]~b[4][4]。

#### (2) 数组赋初值

```
int a[10]={1,2,3,4,5,6,7,8,9,10};  
int b[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
```

初值个数可以少于数组长度, 系统将从下标为 0 的元素开始对应赋值, 未赋初值的自动置为 0。

注意: 当初值个数少于数组时, 有些编译系统会警告个数不匹配。

#### (3) 数组作为参数传递

由于数组名实际上是一个指针, 所以把数组名当作参数传递, 实际上是传地址, 被调用函数可以改变数组元素的值。

**【例 12-13】** 有如下程序, 这是一个排序函数, 它将 aa 数组中的数据从大到小排序。

```
#include <stdio.h>  
void sort(int a[], int n)  
{ int i,j,t;  
  for (i=0; i<n-1; i++)
```

```
        for (j=i+1; j<n; j++)
            if (a[i] < a[j]) {t=a[i]; a[i] = a[j]; a[j] = t; }
    }
void main( )
{   int aa[10]={1,2,3,4,5,6,7,8,9,10}, i;
    sort(aa, 10);
    for (i=0; i<10; i++) printf(" %d, ", aa[i]);
    printf("\n");
}
```

运行结果是：10, 9, 8, 7, 6, 5, 4, 3, 2, 1,

可以看到，sort( )函数改变了aa数组的值。还有一种情况就是把一个数组作为参数传递给函数时，不一定必须从数组的第一个元素地址开始，也可以从其中某个元素开始，取这个元素的地址作为起始地址传递给函数，则函数数组的第一个元素就对应这个元素。比如，若上面例子中的调用改为：sort(&aa[3], 5); 即要求从第4个元素开始的5个元素排序。

#### (4) 数组作为参数传递时的说明

一维数组作为参数传递时，函数头的虚参说明时，下标可以省略，例如：

```
void sort(int a[ ], int n)
```

但二维以上数组只能省略一维，即省略第一维，例如：

```
float f(int b[ ][N])
```

是正确的，但 float f(int b[ ][ ])是错误的；

```
float f(int c[ ][M][N])
```

是正确的，但 float f(int c[ ][ ][ ])是错误的。

原因是数组在实际存放时，多维数组向一维内存映射时需要后面的长度说明。若不将后面几维数组的长度传给被调用的函数，被调用函数将无法确定映射关系。

由于要事先确定后几维数组的实际长度，因此使用多维数组作为参数的函数往往不具有通用性，不能作为通用函数库的函数。

### 12.1.21 指针

#### (1) 指针的类型

若有以下定义：

```
char *p, a, c[10]; double *q, b, d[20];
```

则 `sizeof(a)` 为 1, 而 `sizeof(b)` 为 8, 但 `sizeof(p)` 和 `sizeof(q)` 都为 4, 说明无论是什么类型的指针, 都是占 4 个字节的内存, 指针的类型只是让系统知道一些实际操作中要考虑的类型, 比如同样是 `p++`, 即指针加 1, 对于 `char` 型, 指针是加 1 (1 个字节), 但对于 `double` 型, 指针是加 8 (8 个字节)。

指针只是存放地址, 在未指向一个实际的内存单元之前, 都是不能使用的。

例如: `p = &a;` 或 `q = (double *)malloc(sizeof(double) * 10);`

前者是指向变量 `a`, 后者是动态分配 10 个双精度单元给 `q`。下面就可以对 `p`、`q` 所指向的单元进行操作, 例如:

`*p = 's';` 或 `p[0] = 's';` 两者等价。

`*(q+5) = 1.2;` 或 `q[5] = 1.2;` 两者等价。

由此可以看出, 指针类型和数组的一致性, 但数组名是一个固定的指针。

`*p = 's';` 和 `a = 's';` 两者等价, 因为它们是同一个内存单元。

## (2) 动态分配内存

下面的程序段:

```
char *p; float *q;  
p = (char *)malloc(sizeof(char) * 10);  
q = (float *)malloc(sizeof(float) * 20);
```

执行时, 动态分配 10 个字符单元给 `p`, 20 个浮点单元给 `q`。但要注意使用时, 在程序的开头需要加上文件引用说明

```
#include <malloc.h>  
或 #include <stdlib.h>
```

## (3) `*r++` 和 `(*r)++` 的区别

`*r++` 是先取 `r` 所指单元的内容, `r` 指针再加 1 (指向下一个);

`(*r)++` 是先取 `r` 所指单元的内容, 再将 `r` 指针所指单元内容加 1, 指针并未移动。

## (4) 函数之间传地址

由于 C 语言函数之间参数传递是传值, 为了将被调用函数的结果通过参数传回给调用它的函数, 需要传地址给函数。而函数是按指针来对这些参数进行操作。

## (5) 指针数组

若有下面的程序段:

```
int *a[10], b, c;  
a[0] = &b; a[1] = &c;  
a[2] = (int *)malloc(sizeof(int) * 5);  
*a[0] = 5;  
a[1][0] = 10; /* 与 *a[1] = 10 等价 */
```

```
*(a[2]+2) = 20;  
a[2][3] = 30; /* 与 *(a[2]+3) = 20 等价 */
```

其中 `a` 是指针数组, 有 10 个指针单元, 每一个指针单元都可以存放一个整型变量或内存区的地址。上面的例子中, `a[0]` 指向了 `b`, `a[1]` 指向了 `c`, 而 `a[2]` 则指向了动态分配的长度为 5 个 `int` 型的内存区。

`*a[0]=5` 等价于 `b=5`, `a[1][0]=10` 等价于 `c=10`, `*(a[2]+2) = 20` 是给动态分配的 5 个单元的第 3 个赋值为 20, 而 `a[2][3] = 30` 是给动态分配的 5 个单元的第 4 个赋值为 30。

可以留意一下, 命令行参数 `argv` 就是一个指针数组。

```
void main(int argc, char *argv[ ])
```

(6) 关于 `int (*a)[10]`

若有说明 `int (*a)[10]`, 则 `a` 是一个指针单元, `a` 后面跟的 `[10]` 只是一个长度声明, 可以用 `a` 来动态分配一个二维数组, 后面的 10 是系统存放时, 二维数组向一维内存映射时需要的长度说明。

```
a = (int (*)[10]) malloc(sizeof(int) * 100);
```

系统分配 100 个 `int` 内存单元给 `a`, 构成一个 `10×10` 的二维数组, 引用单元时可以使用 `a[5][5]`, 把 `a` 当作一个静态定义的数组来使用。

### 12.1.22 字符串

**【例 12-14】** 下面的程序给字符串赋初值。

```
main( )  
{ char *p= "OK!"; char a[10]= "OK!"; char b[]="OK!"; }
```

字符串可以通过赋初值的形式赋值。`p` 指向长度为 4 存放 "OK!" 字符串的内存区; 字符串 "OK!" 将赋值到 `a` 数组的单元中, 但 `a` 数组的长度仍为 10; 由于字符串 "OK!" 将在内存中占 4 个字节, `b` 数组的长度将自动定义为 4, 并将字符串 "OK!" 赋值到 `b` 数组的单元中。但不能出现下面的情况:

```
main() { char a[10]; a = "OK!"; }
```

即不能用赋值语句直接进行字符串赋值, 要想实现此功能, 只能使用 `strcpy` 这样的函数, 即: `strcpy(a, "OK!");`

在 C 语言的库函数中, 有一组函数在 `string.h` 中, 实现各种字符串的操作, 下面介绍几个常用的函数:

- `strlen(char *s)` 求字符串 `s` 的长度。
- `strcpy(char *a, char *b)` 将 `b` 串复制到 `a` 所指单元。
- `strncpy(char *a, char *b, int n)` 将 `b` 串前 `n` 个字符复制到 `a` 所指单元。
- `strcat(char *a, char *b)` 将 `b` 串复制连接到 `a` 串的串尾上。



- `strcmp(char *a, char *b)` 比较 a 串和 b 串的大小（按字典顺序）：当 a 串和 b 串相等时返回值为 0；当 a 串大于 b 串时返回值为 >0；当 a 串小于 b 串时返回值为 <0。
- `char *strstr(char *a, char *b)` 确定 b 串是否在 a 串中作为子串出现过，是则返回首次出现位置的地址，否则返回值为 NULL。

### 12.1.23 函数的进一步讨论

#### (1) 命令行

执行一个程序 `ex.exe` 方式是：

```
c:> ex 1.dat 100
```

这里 `1.dat` 和 `100` 就是命令行参数，

C 主函数的参数，就是传递命令行参数：

```
main(int *argc, char *argv[])
```

上面的执行命令使 `argc` 为 3（包括程序名本身），指针 `argv[0]` 指向字符串 "ex"，`argv[1]` 指向字符串 "1.dat"，`argv[2]` 指向字符串 "100"。

要想把 `argv[2]` 所指的数字字符串当整数使用，必须用 `atoi` 函数（`atof` 转换浮点字符串为浮点数）进行转换，即：

```
n = atoi(argv[2]);
```

则 `n` 为 100。

#### (2) 指向函数的指针

对于以下的程序段：

```
float f1(int a) {...}
float f2(int b) {...}
main( )
{   float (*fp)( ), y;
    fp = f1;    y = (*fp)(10);    /* 调用函数 f1 */
    fp = f2; y = fp(20);          /* 调用函数 f2 */
}
```

其中 `float (*fp)( )` 就是定义 `fp` 为函数指针，它可以指向一个函数名。调用方式 `(*fp)(10)` 和 `fp(10)` 是等价的。这里要注意的是，不要写成 `fp = &f1`，因为函数名本身就是一个指针。

#### (3) 函数的递归调用

C 函数可以直接或间接调用函数本身。但必须是有条件的调用，否则就会形成死循环。

**【例 12-15】** 对于菲波那级数： $f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2)$ 。递归程序为：

```
int f(int n)
{
    if (n== 0) return 0;
    else if (n== 1) return 1;
    else return f(n-1) + f(n-2);
}
```

写递归程序的关键是利用归纳法总结出问题的递归式。得到递归式后，加上初始条件，就可以编写递归函数。有兴趣的话，读者可以首先思考下面的问题，看能否归纳总结出递归式，并编写程序。

问题：找出从自然数 1, 2, 3, ..., n 中任取 k 个数的所有组合,如 n=5, k=3。

### 12.1.24 作用域和存储类型

C 语言变量和函数的作用域与存储类型紧密相关。存储类型如下所述。

- **register**: 寄存器类型，主要是为说明整型循环变量，可以提高执行效率；但由于计算机的寄存器有限，此类变量不能定义太多。
- **auto**: 自动存储类型，局部变量都默认为 auto 类型，通常 auto 都不写。
- **static**: 静态存储类型，编译时静态分配内存赋初值，在程序开始执行后，无论其所在函数是否被执行到，也不再释放不再赋初值。
- **extern**: 外部引用，声明此变量说明只是引用，而不是定义。

【例 12-16】 有程序

```
# include <stdio.h>
int a; /* 外部变量 */
main()
{
    int a=0; /* 局部变量 */
    printf("%d\n", f(5) ); printf("%d\n", f(10) );
}
int f(int n)
{
    static int a=0; /* 静态变量 */
    int b=0; /* 局部变量 */
    a += n; b += a;
    return b;
}
```

程序的执行结果是：5, 15

注意：

- 外部变量 a 在所有函数中都可以访问，但如果函数中有同名的局部变量或静态局

部变量, 将掩蔽外部变量, 即函数中只能看到局部变量 (C++中用::a 方式访问同名外部变量)。

- 函数中局部变量的赋初值, 在每次调用时都进行一次, 例如, 函数 f 中的 int b=0; 在 f(5)和 f(10)两次调用时, 都重新赋值为 0, 没有累计效应。
- 静态变量只在程序开始运行时赋一次初值, 并且内存单元一直不释放, 与是否调用该函数无关。所以函数 f 的 static int a=0;只在启动时赋一次初值, 当第一次函数调用 f(5)后, a 的值为 5, 函数返回值为 5; 当第二次函数调用 f(10)后, a 的值为 15, 函数返回值为 15, 有累计效应。

### 12.1.25 结构与联合

#### (1) 结构体

C 语言的结构体的关键字是 struct, 而联合体的关键字是 union。它们都是将不同类型的数据复合成一个整体, 便于进行读、写、赋值、作为参数传递等操作, 而且更便于描述现实世界的实体。

**【例 12-17】** 有以下程序说明

```
struct student {  
    char name[10];      /* 姓名 */  
    long sno;           /* 学号 */  
    char sex;           /* 性别 */  
    float score[4];     /* 四门课的成绩 */  
} *p, a, b;
```

则 sizeof(struct student)为 31。

如果 p = &a; 则 a.sex= 'f' 与 (\*p).sex= 'f' 与 p->sex= 'f'与 p[0].sex= 'f'是完全等价的。

而 b=a 或 b= \*p; 则是将一个结构体 a 的内容赋值给另一个结构体 b。即结构体可以整体赋值。

函数之间可以传递结构体, 但也只是单向传值。

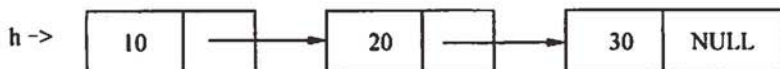
可以利用结构体的复合类型以及动态内存分配来构成链表、树、图等数据结构。

**【例 12-18】** 有以下程序

```
struct NODE {  
    int data;  
    struct NODE *next;  
} *p, *q, *h;  
h = (struct NODE *)malloc(sizeof( struct NODE));
```

```
p = (struct NODE *)malloc(sizeof( struct NODE));
q = (struct NODE *)malloc(sizeof( struct NODE));
h->data = 10; h->next = p;
p->data = 20; p->next = q;
q->data = 30; q->next = NULL;
```

构成一个如下的单链表，头指针为 h。



## (2) 联合体

联合体 union 和 struct 的定义和使用是相同的。但 union 中的各个数据之间共享起始地址相同的单元，所占内存单元的大小是几个数据项中长度最大的一个。

**【例 12-19】** 有以下程序

```
union student {
    char name[10];
    long sno;
    char sex;
    float score[4];
} *p, a, b;
```

则 sizeof(union student) 为 16，长度是最长一项 float score[4] 的长度。

由于数据是共享内存，而不是独立占有各自的内存，所以使用时，要注意赋值和使用的时序。若：

```
a.sno=12345; a.sex='f';
```

则这时 a.sno 的值已经不是 12345。

## (3) typedef

自定义类型 typedef 只是为了用户书写程序的方便，把书写复杂的类型另取一个别名，这也可以使程序简洁易读。

**【例 12-20】** 有以下程序

```
struct student {
    char name[10];
    long sno;
    char sex;
    float score[4];
};
```

定义后，可以用这个 struct student 类型定义变量，申请动态内存：



```
struct student *p, a;  
p = (struct student *)malloc(sizeof( struct student));
```

但如果首先定义:

```
typedef struct student STU;
```

则上面的两个语句就可以书写成:

```
STU *p, a; p = (STU *)malloc(sizeof(STU));
```

使书写简单并且易读。但 typedef 与宏定义一样,这只是一符号的代换,并不能提高程序的效率。

### 12.1.26 位运算

C 语言的位操作是针对整数类型(包括: char, int, short, long 类型)而设立的,运算符有:

~(求反)、<<(左移位)、>>(右移位)、&(按位与)、^(异或)、|(按位或)。

注意:

- 求反~不同于逻辑操作的非操作!。例如: char a=0xA0, 其二进制是 10100000, 则~a 结果为 01011111, 即 0x5F; 而!a 是当 a 的值为 0, 则结果为 1, 而 a 的值为非 0 时结果为 0, 所以这里!a 的值为 0。
- <<(左移位)和>>(右移位)经常用于整数乘除 2 的整数倍数, 可以提高执行效率。例如: char a=21, 则 a<<1 的结果为 42, 而 a>>1 的结果为 10。
- &(按位与)、^(异或)经常用于检测标记变量中某位的值是否为 1, 或将某一位强置为 1。例如: 若 a 是 char 型, if(a & 0x80) 就是检测 a 的最高位(第 8 位)是否为 1, 是为真。而 a=a|0x80; 则是将 a 的最高位(第 8 位)强置为 1。
- ~(求反)为一元操作符, 运算优先级高于所有二元运算符, 而与其他一元运算符相同。其他 5 个运算符<<、>>、&、^、| 的运算优先级是:
  - (a) <<、>> 低于算术运算符+、-, 但高于关系运算符;
  - (b) & 低于关系运算符, 但高于^, ^高于|, |高于&&(逻辑与)。
- <<、>>、&、^、| 可以和赋值号=结合, 形成复合赋值运算符: <<=、>>=、&=、^=、|=。例如: a<<=2 与 a=a<<2 是等价的。

### 12.1.27 文件操作

C 语言的文件操作是通过文件指针和一族函数实现的。使用的大体步骤是:

(a) 要进行文件操作, 首先要在文件头部加上文件引用说明

```
#include <stdio.h>
```

(b) 用 FILE 类型定义文件指针

FILE \*fp;

(c) 用 fopen 函数打开文件

fp=fopen(<文件名>, <打开方式>);

其中:

<文件名>是要进行操作的文件;

<打开方式>有多种, 常用的有:

- “r” 文本只读方式。
- “rb” 二进制只读方式。
- “w” 文本写方式, 创建新文件, 不管原来文件是否存在。
- “wb” 文本写方式, 创建新文件, 不管原来文件是否存在。
- “r+” 文本读写方式: (文件打开前应该存在)。
- “w+” 文本读写方式, 先创建文件, 不管原来文件是否存在, 然后可以移动文件指针, 读文件。
- “r+b” 二进制读写方式 (文件打开前应该存在)。
- “w+b” 二进制读写方式 (先创建文件, 不管原来文件是否存在)。
- “a” 文本添加方式, 若文件不存在, 则创建新文件; 若文件存在, 则在文件尾部添加内容, 不会破坏原文件的内容。
- “ab” 二进制添加方式, 若文件不存在, 则创建新文件; 若文件存在, 则在文件尾部添加内容, 不会破坏原文件的内容。

(d) 若是用文本方式打开的文件, 则用 fscanf(), fprintf(), fgetc(), fputc(), fgets(), fputs() 等进行读写, 后 4 个函数只用来读写字符型; 而若是用二进制方式打开的文件, 则用 fread(), fwrite() 等进行读写。

(e) 可以利用函数 feof() 判断是否读到了文件尾, 当未读到文件尾时, feof(fp) 的返回值是 0, 通常用 while (!feof(fp)) 来构成读循环。

(f) 若是用二进制方式打开的文件, 可以使用 fseek() 函数来移动文件指针的位置, 以便随机读写。

(g) 可以用 ftell(fp) 取得文件流 fp 的当前读/写位置 (相对于文件头的偏移量)。

(h) 用 fopen() 打开一个文件, 必须对应地使用一个 fclose() 关闭该文件, 以释放文件操作软通道 (个数是很有限的), 便于下面继续使用。

关于文件操作函数的使用细节, 这里不多介绍, 可以查阅相应的 C 参考手册。

### 12.1.28 C++简介

C++是目前在实际应用中使用非常广泛的一种语言。C++最初被称为多类 C (C with Class), 是 1983 年由美国 AT&T 公司新泽西贝尔实验室的 Bjarne Stroustrup (现在是美国德克萨斯 A&M 大学的教授, 个人主页: <http://www.research.att.com/~bs/>) 在 C 语言的

基础上研制出来的。目的是用于大型软件（指源程序超过 5 万行）的开发。主要是为了解决 C 语言程序不受任何限制而导致的副作用严重的问题。比如 C 程序中的一个外部变量，程序中的任何函数通常都可以访问，出现数值异常问题经常难以马上确定是哪个函数造成的。因此 C++ 增加了某些机制，允许程序员生成严格控制存取权限的“黑盒子功能单元”——对象（object），因此 C++ 被称为面向对象的程序设计语言。C 是 C++ 的子集，C++ 保持了 C 的原始思想，程序员写程序时可用也可以不用 C++ 所增加的功能。

C++ 在 C 的关键字的基础上增加了：class, delete, friend, inline, overload, operator, new, public, this, virtual。对于某些特定的 C++ 编译器，如 Visual C++ 6.0 编译器，还会自己定义一些新的关键字。

C++ 增加的最重要的机制有三个：类(class)、函数重载(function overloading)、操作符重载(operator overloading)。

### 12.1.29 关于 C++ 的几个基本问题

#### (1) 参数的说明

C++ 鼓励对函数参数返回值及参数全面说明，并且在函数名的括号中说明参数类型。并希望向前引用说明也不只是对函数名类型说明，而是全面对参数说明，这有利于编译系统对函数调用的实参类型进行全面类型检查，及早找出错误。

例如，C 语言中：float fun(a, b, c)

float a, b, c;

C++ 写成：float fun(float a, float b, float c)

#### (2) 注释

C 语言用 “/\*” 和 “\*/” 进行注释，从 “/\*” 开始一直到 “\*/” 为注释信息。而 C++ 程序中增加了行注释：“//”，从 “//” 开始到本行尾（回车）为注释信息。

例如：void main()

```
{ int i; // i 是循环变量
  float sum; // sum 用来求累加和
  ...
}
```

#### (3) 内联函数 (inline)

为了提高执行效率，C++ 提供了内联函数功能，与宏定义 define 类似，编译器将出现函数名的位置用函数体进行替换，省去了真正函数调用的压栈退栈时间。内联函数比 define 方便，功能也更强，不只是简单的字符串的替换，也考虑了类型匹配问题。

内联函数的书写有一些限制：内联函数中不能使用循环语句和 switch 语句，也不能进行异常接口声明(throw)等。

#### (4) 输入/输出

C++用 cin、cout 和其他一些新引入的函数进行输入/输出, 但需要引用文件 iostream.h。

例如: cin >> i >> f; // 等同于 scanf("%d%f", &i, &f);  
cout << i << f << endl; // 等同于 printf("%d%f\n", i, f); 其中 endl 等同于 '\n'

#### (5) 动态内存分配

C++用 new 和 delete 动态分配和释放内存, 等同于 C 语言的 malloc 和 free 函数。

例如: int \*p=new int(10); // 分配 1 个 int 单元, 并赋初值为 10;  
// 等同于 C 的: int \*p=(int \*)malloc(sizeof(int)); \*p = 10;  
int \*p=new int[10]; // 分配 10 个 int 单元, 即 pt 是一个数组, pt[0]~pt[9]  
// 等同于 C 的: int \*p=(int \*)malloc(sizeof(int)\*10);  
delete p; // 释放 p 所指的 1 个 int 单元  
delete [ ]pt; // 释放 pt 所指的 10 个连续 int 单元

#### (6) 指针/地址的传递

C 的参数传递只是传值, 无法将变量地址传递到函数中。而 C++提供了传递变量地址这种机制, 方法是: 在定义函数时在要传递地址的变量前加 &, 使用时跟普通调用一样。

#### (7) 类型修饰符 const

const 是类型修饰符, 在所定义的变量类型前面或后面出现, 例如, int const a=1; 或 const int a=1; 使变量 a 成为不可修改的量, 等同于常量。但 const 比 C 语言的常量功能更强, 因为 const 是带类型的。而且 const 经常用于函数参数说明, 防止在函数中无意地破坏参数的值或所指内容。如 strcpy(char \*str2, const char \*str1); 的功能是将字符串 str1 复制到 str2。将参数 str1 定义为 const, 就是防止在 strcpy 函数中破坏了 str1 所指的字符串。

#### (8) 作用域与可见性

在 C 语言中可以出现局部变量和全局变量同名, 但函数中的局部变量将掩蔽全局变量, 直到局部变量的作用域失效才能访问同名的全局变量, 而实际应用中可能需要同时访问同名的局部变量和全局变量。C++为了克服这个缺点, 引入了作用域操作符 (scope operator) '::', 使我们可以同时访问同名的局部变量和全局变量。

#### (9) 默认参数

在函数声明中可以为一个或多个参数指定默认值, 这样函数调用时可以从右边开始省略一个或多个参数。

### 12.1.30 类

类 (class) 是抽象数据类型的实现。类不是类型, 是将数据和相应的对这些数据的操作函数的封装, 形成对象 (object)。class 不同于 struct、union, 后者是数据类型, 但其中不包括函数 (操作)。

类可以继承和派生, 并引出了访问权限控制和调整、保护成员、成员函数名静态和



动态束定（虚函数）等问题，这里限于篇幅我们不一一介绍，只讲一些最基本的内容。

### (1) 类的定义

【例 12-21】 有一个日期类 Tdate 定义如下：

```
class Tdate
{
public:
    void Set(int d, int m, int y);
    int IsLeapYear( );
    void Print( );
private:
    int day, month, year;
};

void Tdate::Set(int d, int m, int y)
{ day=d; month=m; year = y; }

int Tdate::IsLeapYear( )
{ return (year %4 == 0 && year%100 != 0) || (year%400) == 0); }

int Tdate::Print( )
{ cout <<month<< "："<<day<< "："<<year<<endl; }

int main( )
{ Tdate Today, Tomorrow; //定义了两个对象
  Today.Set(24, 11, 2004);
  Tomorrow.Set(25, 11, 2004);
  if (Today. IsLeapYear( ))
      cout << "Is Leap Year!";
}
```

其中类 Tdate 包含数据 day, month, year 和函数 Set、IsLeapYear、Print，函数 Set、IsLeapYear、Print 被称为是成员函数，是 Tdate 类的公有部分，其他函数可以调用成员函数。而数据 day, month, year 是 Tdate 类的私有部分，仅成员函数 Set、IsLeapYear、Print 可以访问，其他函数不可访问。即使是在 main 函数中也不能出现 Today.month=1。也就是说，其他函数要访问类的私有部分，必须通过其成员函数间接访问。

成员函数说明时要加上一个类的前缀 Tdate::，表明是这个类的成员函数。

利用类可以定义一个或多个对象，上面的例子就用 Tdate 类定义了两个对象 Today 和 Tomorrow，对象之间无论是数据还是成员函数都是独立的。

### (2) 构造函数/析构函数

一个类建立后，往往需要初始化，例如申请动态内存，程序结束时释放。若忘记会出现问题。C++ 提供了构造函数/析构函数：与类同名的成员函数被称为是构造函数 (constructor)，而与类同名带前缀 '~' 的成员函数被称为是析构函数 (destructor)，它们不在程序中直接调用，而是在对象创建和撤销时自动调用。

若程序中没有定义构造函数/析构函数,则编译器自动生成一个这样的函数,叫做公有构造函数/析构函数。

### (3) 静态成员

利用类定义的对象之间是相互独立的,若想在多个对象之间共享数据,则可以用静态成员。

**【例 12-22】** 若定义了

```
class MyClass {  
    public:  
        MyClass(int i);  
        ~MyClass( );  
        void Print( );  
    private:  
        int member;  
        static int global;  
};
```

则无论定义多少个 MyClass 对象,静态成员 global 在整个程序中只有一个。不同 MyClass 对象访问的是同一个 global。

若一个成员函数只存取静态成员,则这个函数应该声明为静态成员函数。静态成员函数在整个程序中也将只有一个。

### (4) 友元

C++ 提供了友元机制,放松对于对象私有成员访问的严格限制,让不是成员函数的其他函数也能访问私有成员,这有时会方便编写程序。但每一个友元都必须在类中声明,这有利于对访问权限的控制。

## 12.1.31 函数重载

C 语言中不允许出现函数重名,这给程序员编写程序有时会带来麻烦。比如,对不同类型的变量求绝对值的函数,若允许同名,程序员只要记住 abs 既可,防止使用 fabs、labs 等出现的混乱。C++ 提供了这种机制,只要同名的函数的参数个数和参数类型不完全相同即可,这样编译器就可以通过调用函数的参数个数和类型来静态绑定相应的函数。

## 12.1.32 操作符重载

C++ 允许将原来的操作符加入新的含义,方便程序员编程。但操作符必须是已有的,不能自己定义新的操作符,可重载的操作符有 45 个,都是一目、二目操作符。并需要为操作符新加入的每一个功能编写一个操作符函数,而且只能是类类型的重载操作符。步

骤是:

- ① 生成一个类, 定义重载操作符的操作数据类型;
- ② 在 `public` 中, 包含一个友元函数或一个操作符成员函数;
- ③ 在程序中, 定义一个操作符函数。

### 12.1.33 类的继承和派生

继承和派生(Derivation and Inheritance)是 C++ 的重要机制, 是面向对象设计的重要特征, 该机制自动为一个类提供来自另一个类的数据结构和操作。这样可以利用已构造好的类生成新的类, 充分利用已有资源。

#### (1) 概述

类继承的语法形式如下所述。

`class` 标识符 2: 访问控制 标识符 1

{ 新成员和成员函数 }

其中: 标识符 1 为基类; 标识符 2 为派生类; 访问控制是指类的继承方式, 有三种: `public`(公有继承)、`protected`(保护继承)、`private`(私有继承); 新成员和成员函数是指在继承基类的成员和成员函数的基础上, 新增加的成员和成员函数。

【例 12-23】 点类与矩形类:

```
class Shape {
    public: void SetB(int x, int y);
           void MoveB( );
           void DisplayB( ) const;
    private: int X, Y;
};

class Rectangle: public Shape {
    public: void SetD(int x, int y, int w, int h);
           void DisplayD( ) const;
    private: int Width, Height;
};
```

#### (2) 访问控制 (继承性质)

当继承性质为 `public` 时, 则派生类全部继承基类的成员和成员函数, 包括访问权限; 当继承性质为 `private` 时, 则禁止派生类对基类的直接访问, 即使是派生类的成员函数也不能直接访问基类的成员和成员函数。派生类的成员函数只能访问基类中具有公有访问特性的成员和成员函数。通过不同性质的继承, 派生类改变了对基类的访问权限, 例如对于上例的成员函数:

```
void Shape::SetB(int x, int y)
```



```
{ X=x; Y=y; }  
void Rectangle::SetD(int x, int y, int w, int h)  
{ SetB(x, y); // 不可以 X=x, Y=y;  
  Width = w;  
  Height = h;  
}
```

虽然 Rectangle 类通过 public 方式继承了 Shape 类的所有成员和成员函数, Shape 类的私有成员 X, Y 也是 Rectangle 类的私有成员, 但 Rectangle 类的成员函数 SetD 仍不能直接访问 X, Y, 而要通过 Shape 类的公有成员函数 SetB 间接访问。

### (3) 保护成员

保护成员是指 protected 声明的成员, 保护成员具有双重身份: (1) 对于建立它的类, 它与 private 成员的性质相同; (2) 对于基于此类的派生类, 它与 public 成员性质相同。

#### 【例 12-24】 保护成员。

```
class MyClass {  
    public: void Print( ); int p;  
           void Set( );  
    protected: int val;  
    private: int X, Y;  
};  
void main( )  
{ MyClass My;  
  My.Set( );  
  My.p=5;  
  My.val = 10;  
  My.Print( );  
}
```

语句 My.val = 10; 编译时系统会给出错误信息, 说明 val 不允许直接访问, 只有 MyClass 类的成员函数可以访问, 所以保护成员 val 的性质等同于私有成员。

#### 【例 12-25】 派生类与保护成员。

```
class Yourclass : public MyClass {  
    public: void Update(int x, int y, int v);  
};  
void Yourclass :: Update(int x, int y, int v)  
{ Set(x, y);  
  val = v; // Yourclass 类可以直接访问 val, 但不能直接访问 X, Y  
}
```



```

void main( )
{   Yourclass You; You.Update(3,5,7);
    You.Print( );
    You.val=9; // 编译给出错误信息
}

```

在 Yourclass 类的成员函数 Update 中, 可以出现 `val = v`; 说明 Yourclass 类可以直接访问基类 Myclass 的保护成员 val, 但不能直接访问基类的私有成员 X,Y。但在 main 函数中出现 `You.val=9`; 编译时系统仍会给出错误信息, 这说明 val 仍不是 Yourclass 真正的 public 成员, 只是 Yourclass 的 protected 成员, 性质同 Yourclass 类的 private 成员。

归纳总结一下: 若 A 是基类, B 是基于 A 的 public 派生类, 则: (1) A 的 private 成员, 是 B 的 private 成员, 但 B 的成员函数不能直接访问, 仍然是 A 的 private 成员, 但 A 是 B 的一部分; (2) A 的 protected 成员 (函数), 是 B 的 protected 成员 (函数), 与 B 的 protected 成员 (函数) 融和成一体。实际上成为 B 的 private 部分; (3) A 的 public 成员 (函数), 是 B 的 public 成员 (函数), 与 B 的 public 成员 (函数) 融和成一体。

#### (4) 友元与继承

由于派生类不能直接访问基类中的私有成员, 所以若一个派生类要直接访问基类中的私有成员, 可以将这个派生类声明为基类的友元。

#### 【例 12-26】友元。

```

class Myclass {
{   friend class Yourclass;
    public: void Set(int x, int y);
    private: int X, Y;
}
class Yourclass : public Myclass {
    public: void Set(int x, int y, int val);
           void Print( );
    private: val;
}
void Myclass::Set(int x, int y)
{   X=x; Y=y; }
void Yourclass::Set(int x, int y, int v)
{   X=x; Y=y;           // 可以直接访问基类的私有成员 X,Y
    Myclass::Set(x, y); // 也可以调用基类的公有函数
    val=v;
}
void Yourclass::Print( )
{   cout <<X <<' ' <<Y <<' ' <<val <<endl;}

```

```
void main( )
{ Yourclass Obj;
  Obj.Set(1, 3, 5);
  Obj.Print( );
}
```

### (5) 访问权限调整

在私有继承的情况下，基类的成员或成员函数在派生类中具有私有成员的特性。通过访问权限调整，可恢复派生类所继承的成员或成员函数在基类中的访问权限。但只能恢复原状，不可能把 private、protected 成员调整为 public。

#### 【例 12-27】 访问权限调整。

```
class Myclass {
public: int f( );
      void f(int i);
      void g( );
protected: void h( );
private: int X;
};

class Yourclass : private Myclass {
public: void k( );
      void hh( );
      Myclass::f;      // 恢复 f 的公有属性
      Myclass::g;      // 恢复 g 的公有属性
protected: Myclass::h; // 恢复 h 的保护属性
};

int Myclass::f( )
{ return X; }

void Myclass::f(int i)
{ X=i; }

void Myclass::g( )
{ cout <<X <<endl; }

void Myclass::h( )
{ X++; }

void Yourclass::k( )
{ f( f( ) - 1); } // k 不能直接访问 X，只能调用基类的函数间接访问

void Yourclass::hh( )
{ h( ); } // h 是保护函数不能直接访问，只能由类的成员函数访问

void main( )
{ Yourclass b;
  b.f(5); b.g( ); // 初值置为 5
```

```

    b.hh( ); b.g( ); // 加1    写 b.h( );是不可以的
    b.k( ); b.g( ); // 减1
}

```

### (6) 成员名限定

在派生类和基类中可以声明同名的成员(函数), 不特别说明, 使用时编译器会默认为派生类中的(后出现的), 除非用:

基类名::基类成员名

进行成员名限定, 才能调用同名的基类成员(函数)。

#### 【例 12-28】 成员名限定。

```

class MyClass {
    public: MyClass(int x);
           void Print( );
    private: int X;
};

class Yourclass : public MyClass {
    public: Yourclass(int x, int y);
           void Print( ); // 派生类的同名成员函数
    private: int X;      // 派生类的同名私有成员
};

void MyClass::Print( ) // 基类的 Print 函数
{ cout << X << " base class\n"; }

void Yourclass::Print( ) // 派生类的 Print 函数
{ MyClass::Print( );     // 派生类调用基类的 Print 必须加前缀 MyClass::,
                          // 否则调用成为自递归
  cout << X << " derivation class\n";
}

MyClass::MyClass(int x)
{ X=x; }

Yourclass::Yourclass(int x, int y):MyClass(x)
{ X=y; } // 注意: 这里不能直接函数调用 MyClass(x)

void main( )
{   Yourclass r(3,5);
    r.Print( ); // 调用派生类的 Print 函数
    r.MyClass::Print( ); // 调用基类的 Print 函数
}

```

### (7) 动态束定与虚函数

virtual 所修饰的成员函数, 称为虚函数。而程序运行时进行的束定称为动态束定。当一个类是从一个或多个类派生出来时, 往往会出现多个同名且同参数个数和参数类型

的成员函数，分别实现不同的功能。但这些函数不是重载函数（重载函数必须是参数个数或参数类型不同，可以静态束定），编译器无法静态束定到底是要调用哪一个成员函数，延迟到程序运行时再进行动态束定，由 `this` 指针决定调用哪一个同名的成员函数。需要在重名的成员函数前加上 `virtual` 声明。此种类被称为多态类。

**【例 12-29】 虚函数。**

```
# include <iostream.h>
class Shape { // 点类(X,Y)
public: Shape(double x, double y);
       virtual double Area( ) const;
private:double X,Y;
};
Shape::Shape(double x, double y):X(x),Y(y){ }
double Shape::Area( ) const
{ return 0.0; }
class Circle : public Shape { // 圆类 = 点类+半径
public: Circle(double x, double y, double r);
       virtual double Area( ) const;
private:double Radius;
};
```

在派生类中重定义虚函数时，必须满足：

- ① 与基类的虚函数参数个数相同；
- ② 与基类的虚函数参数一一对应；
- ③ 函数返回类型或与基类的虚函数相同，或都返回指针或引用（可以不同类型）。

若满足上述三条件，有时可以省略 `virtual` 关键字，编译器会自动认为是虚函数。但从可读性出发应该保留 `virtual` 关键字。

**(8) 多继承**

一个派生类可以同时从几个基类派生而来的。派生类将继承所有基类的各种成员和成员函数，并可以加入新的成员和成员函数。当建立派生类的对象时，基类构造函数被执行，派生类的对象中所包含的基类子对象被基类的构造函数初始化。但为了使用基类的某个特定的构造函数初始化派生类对象中所包含的基类子对象，必须在派生类的构造函数的成员初始化列表中显示表示对这个基类构造函数的调用。

### 12.1.34 模板

模板（`template`）是 C++ 支持参数化多态性的工具，是将一段程序所处理对象的类型参数化，这样各种类型的对象都可以用同一段程序处理，省去重复编写程序。我们把这样的一段将类型参数化的程序称为一个模板。C++ 模板主要针对函数和类，形成函数



模板和类模板。

语法形式为：

template < 模板参数表 > 声明

模板参数表：

class 标识符

类型表达式 标识符

这里声明是函数或类的说明或定义，标识符即数据类型。

例如：<class T>

```
T abs(T a)
{ return a<0 ? a: -a; }
```

就是一个求数的绝对值的函数模板，只要 a 是和 0 可以比较的类型，都可以使用这个模板。

### (1) 函数模板

函数模板是为防止为了实现相同功能而不得不写代码相同的多个重载函数。编译系统根据实际调用情况，由模板生成重载的模板函数。使用函数模板只是简化代码书写，并不能提高执行效率。如果一个函数模板声明中定义有局部的静态对象，那么这个静态对象也在每个模板函数中被定义，各模板函数之中的静态变量是相互独立的。

#### 【例 12-30】 函数模板。

```
# include <iostream.h>
template <class T>
T abs(T a)
{ return a<0 ? -a : a;
}
void main()
{ char c=-0x63; int d=-32767;
  float e=-1.2345f; double f=-1.234e10;
  cout <<abs(c) <<' ' <<abs(d) <<' ' ;
  cout <<abs(e) <<' ' <<abs(f) << endl;
}
```

编译时，由 abs(c)、abs(d)、abs(e)、abs(f)这四个函数调用，自动生成四个类型分别为 char、int、float double 的 abs 函数，它们之间是重载的。

若程序中有函数模板 T min(T a, T b)，同时还有同名的函数 int min(int a, int b)和 float min(int a, float b)。则系统按照以下规则处理：

① 编译系统根据实参类型，首先在 min 重载函数中找参数类型匹配的，找到则调用；上例中 min(a,f) 参数是 int 和 float，调用 float min(int a, float b)重载函数。而 min(a,b)

参数是 int 和 int, 调用 int min(int a, int b)重载函数。

② 否则, 试图用函数模板生成一个 min 模板函数, 若能则生成。上例中 min(e, f) 参数是 float 和 float, 使 T=float, 生成 float min(float a, float b)。而 min(d1, d2) 参数是 double 和 double 使 T=double, 生成 double min(double a, double b)。

③ 给出错误信息。上例中 min(f, a) 参数是 float 和 int, 没有匹配的重载函数, 也不能用模板生成, 所以系统给出错误信息。

当多个类型参数化时, 编译系统将对实参的类型进行静态类型分析, 来确定函数模板中对应位置的类型参数的类型。模板函数返回类型参数必须出现在函数声明的形参表中。例如:

```
template <class X, class Y>
Y fun(X x, Y y)
{...}
```

是正确的。而

```
template <class X, class Y, class Z>
Z fun(X x, Y y)
{...}
```

是错误的, 因为编译系统无法确定 Z 的类型。

## (2) 类模板

类模板使实现类所需要的数据类型参数化。适合于包容性质的类。所谓包容性的类, 是指其对象像一个容器, 操作特性与对象类型无关或很少有关系。栈、队列等是典型的包容性质的类, 适合做成类模板。

**【例 12-31】** 栈类模板。

```
template <class T>
class Stack {          // 栈的类模板
public: Stack(int size);
        virtual ~Stack();
        void Push(const T &e);
        const T &Pop();
        const T &Peek() const;
        int IsEmpty() const;
private: T *buff;
        int max;
        int top;
};
template <class T>
```

```

Stack <T> :: Stack(int size)
{   buff = new T[max=size];
    top = -1;
}
template <class T>
Stack <T> :: ~Stack( )
{   delete []buff;
}
template <class T>
void Stack <T> :: Push(const T &e)
{   if (top >= max)
        throw overflow( );
    buff[++top] = e;
}

```

编译系统根据说明: `Stack <double> DoubleStack(20);` `Stack <int> IntStack(50);` 将生成 `double` 型的 `Stack` 对象 `DoubleStack`, 其栈的大小为 20; `int` 型的 `Stack` 对象 `IntStack`, 其栈的大小为 50。每个对象各自包含相应 `double` 和 `int` 类型的成员和成员函数同时生成, 这些函数虽然同名, 但不是重载函数, 因为它们隶属于不同的对象。

在类模板中, 可以对常量进行参数化, 例如:

```

template <class T, class N>
class Array {
    public: T &operator[ ](int i);
    private: T buf[N];
};
template <class T, class N>
T & Array<T, N> :: operator[ ](int i)
{   return buf[i]; }

```

其中, 参数模板 `N` 是参数化的常量, 所参数化的是一个 `int` 常量, 该常量在声明模板类时指定。语句 `Array<int, 20> a;` 从类模板 `Array<T,N>` 中生成模板类 `Array<int, 20>`, 该模板类被用于声明对象 `a`。生成静态的 `int buf[20]`; 而前一个例子的 `buff` 是动态分配的。

### (3) 模板的引用声明与定义声明

仅有类模板名而无类体, 则为引用声明。例如:

```
template <class X, class Y> class MyTemplate;
```

`MyTemplate<X,Y>` 就是一个模板的引用声明。模板的引用声明可用于友元函数和函数模板的函数参数声明。

### 12.1.35 异常处理

在程序执行的过程中若出现一些不平常的情况，或运行结果无法定义的情况，而使得操作不得不被中断时，我们就说程序出现了异常。C++中的异常控制结构，当函数出现异常，执行被终止时，使控制从函数返回，返回点是调用函数所指定的一个地点，而不是调用发生的地点。

语法形式是：

try { 函数调用 }

catch (声明) 语句

被调用函数中使用 throw，当满足条件由 throw 触发异常，而由 try 捕捉其后面 {} 中函数调用出现的异常，throw 的表达式对应 catch 的声明。一个 try 块后可以有多个 catch 处理不同的 throw 异常。

【例 12-32】异常处理。

```
#include <iostream.h>
double Div(double a, double b)
{   if (b == 0.0) throw 1.0E-38;
    return a/b;
}
void main()
{   try {
        cout << "5.2/2.0=" << Div(5.2, 2.0) << endl;
        cout << "5.2/0.0=" << Div(5.2, 0.0) << endl;
        cout << "5.2/4.0=" << Div(5.2, 4.0) << endl;
    }
    catch (double d)
    {   cout << "Exception, float overflow, why->" << d << endl;}
    catch (int i)
    {   cout << "Exception, integer overflow, why->" << i << endl;}
    cout << "All done!" << endl;
}
```

运行结果：

5.2/2.0=2.6

Exception, float overflow, why->1e-038

All done!

由于第二个 Div 函数调用出现异常，转向 catch 处理。所以第三个 Div 函数调用语句没有被执行。



## 12.2 例题分析

### 【例题 12-1】 对于程序

```
main( )
{
    int i1; i1=1000;
    float f1;f1=123.45;
}
```

编译时系统会提示错误信息:

```
line 4: syntax error : missing ';' before 'type'
line 5: 'f1' : undeclared identifier
```

请说明原因。

分析: 这种错误信息可能令你无从琢磨到底是什么错误, 这是因为 float f 的定义出现在赋值语句 i1=1000 之后, 如果改成:

```
main( )
{
    int i1; float f1;
    i1=1000; f1=123.45;
}
```

就不会再有任何错误, 因为变量说明在所有执行语句之前。但下面的程序段:

```
main( )
{
    int i1=1000;
    float f1=123.45;
}
```

编译也不会有错误, 因为给 i1 赋值 1000 是初始化赋值, 不是可执行语句。

### 【例题 12-2】 分析下面程序, i1 的定义和使用是否正确。

```
# include <stdio.h>
main( )
{
    int i1=1000;
    {
        int i1=100;
        printf("i1=%d ", i1);
    }
    printf("i2=%d\n", i1);
}
```

分析：虽然变量 `i1` 在程序中定义了两次，但两次并不是重复定义。因为在任何一个“{”后可以定义新的局部变量，当新定义的局部变量与原有变量同名时，新定义的变量掩盖原有变量。运行结果是：`i1=100 i2=1000`

【例题 12-3】对于下面的程序，请给出输出结果。

```
main() {
    char c1, c2, c3; unsigned char uc1, uc2;
    c1 = 'a'; uc1=97; c3=c1-32; c2 = -97; uc2 = -c2;
    printf("%c %c %c %d %d %c", c1,uc1,c3, uc1, c2, uc2);
}
```

正确答案：`aaA 97 -97 a`

分析：`uc1` 的值为 97，按字符型 `%c` 输出则为字符 'a'，因为字符 'a' 的内码值（ASCII 值）是 97，而 `uc1` 按整型 `%d` 输出则为整数 97。说明一个字符型量在内存中不区分是字符还是整数，是否输出为字符型是由输出格式决定的。

注意：处理中文信息需要使用 `unsigned char` 这种类型，因为 GBK 汉字的编码首字节取值范围是：`0x81~0xFE`，用到了字节的第 8 位。而通常的字符（ASCII 表）取值是 `0~127 (0x00~0x7F)`。

【例题 12-4】对于下面的程序用 Turbo C 2.0（16 位）编译器编译，请给出输出结果。

```
main() {
    short i1, i2;    long l1, l2;
    i1 = 32767; i2=i1+1;
    l1 =32767*32767; l2 = 123456789L;
    printf("%d %d %ld %ld",i1, i2, l1, l2);
}
```

正确答案：`32767 -32768 1 -13035`

分析：`i1` 的值为 32767，`i2` 将 `i1` 加 1，结果应该是 32768，但由于 `short` 类型的取值最大是 32767，所以就发生了溢出现象，结果变成了 -32768。`l1` 的输出结果为 1，可能会出乎许多人的意料，因为 `32767*32767` 的结果为 1073676289，并没有超出 `long` 型最大取值 2147483647，而且输出也是按照长整型 `%ld` 格式。出错的原因是在于语句 `l1=32767*32767`；赋值号右侧相乘的两个数 32767 系统都默认为是 `int` 型，两个 `int` 型数相乘结果也默认为是 `int` 型，取值最大是 32767，因此发生了溢出现象，表达式结果为 1，`l1` 的结果也就是 1。要想得到正确的结果，可以通过两种方法处理：一种是将其中一个或两个数都表示成长整数，即在数的尾部加 `L` 或 `L`，即：

`l1=32767*32767L;`

另一种是先将一个数赋值给 `l1`，再将 `l1` 乘 32767，即：

```
l1=32767; l1=l1*32767;
```

因为 C 语言的计算规则是向“高”类型看齐。所谓“高”类型即是精度高，取值范围大的类型。一个长整数乘一个整数结果是长整数，所以 l1 就可以得到正确的结果。

l2 没有得到预期结果 123456789，而是-13035，不是 l2 的值不对，而是输出格式“%d”不能正确输出长整数 123456789，改为“%ld”格式就会得到正确的结果。

**【例题 12-5】** 对于下面的程序，请给出输出结果。

```
# include <stdio.h>
main()
{   int i, j, n=5; float f1, f2, f3, f4;
    i = 18 % n;
    j = (-18) % n;
    f1 = 1/n;
    f2 = 1.0/n;
    f3 = (float)1/n;
    f4 = (float)(1/n);
    printf("%d %d %f %f %f %f\n", i, j, f1, f2, f3, f4);
}
```

正确答案: 3 -3 0.000000 0.200000 0.200000 0.000000

分析: C 语言的求余%不改变整数的正负号，所以 i 的值为 3，j 的值为-3。f1 值为 0.0 的原因是 1/n 是两个整数相除，结果取整型得 0。f2 值为 0.2 的原因是 1.0/n 是浮点数和整数相除，结果取浮点数得 0.2。f3 值为 0.2 的原因是(float)1 将 1 转化成了浮点类型，因此结果取浮点数得 0.2。f4 值为 0.0 的原因是(float)(1/n)是当 1/n 这两个整数相除结果得 0 后才将其转化成浮点类型，因此结果为 0.0。

**【例题 12-6】** 对于下面的程序，请给出输出结果。

```
# include <stdio.h>
main( )
{   int n=6, m=12, i=1234; float f=12.3456;
    printf("%d ", n, i);
    printf("%*.*f\n", m, n, f);
}
```

正确答案: 1234 12.345600

分析: 在 printf("%d\n", n, i) 语句中，n 是场宽不输出，对应前面的星号，相当于"%6d"格式，因此输出 i 为 1234。而在 printf("%\*.\*f\n", m, n, f) 语句中，m, n 是场宽不输出，分别对应前面的两个星号，相当于"%12.6f" 格式，因此输出 f 为 12.345600。

**【例题 12-7】** 下面的程序试图用 C 语言表示数学上的  $a > b > c$ ，请给出输出结果。

```
# include <stdio.h>
main( )
{   int a=5, b=4, c=3, r1, r2;
    r1 = a>b>c;
    r2 = a>b && b>c;
    printf("%d %d\n", r1, r2);
}
```

正确答案: 0 1

分析:  $r1 = a > b > c$  的表达不正确, 因为  $a=5, b=4, a > b$  为真结果为 1, 而  $1 > c$  为假, 所以最后结果为 0。  $r2 = a > b \ \&\& \ b > c$  是正确的表示, 因为  $a > b$  结果为 1, 而  $b > c$  的结果也为 1, 最终结果为 1。

【例题 12-8】 对于下面的程序, 请给出输出结果。

```
# include <stdio.h>
main( )
{   int c1=0, c2=0, c3=0, c4=0; /* c1, c2, c3 分别是字符 1, 2, 3 的出现次数,
                                c4 是其他的出现次数 */

    char *p="112311224563212";
    while (*p)
    {   switch (*p)
        {case '1': c1++;
         case '2': c2++;
         case '3': c3++;
         default: c4++;
        }
        p++;
    }
    printf("c1=%d c2=%d c3=%d c4=%d\n", c1, c2, c3, c4);
}
```

正确答案:  $c1=5 \ c2=10 \ c3=12 \ c4=15$

分析: 预期的输出结果是  $c1=5 \ c2=5 \ c3=2 \ c4=3$ 。结果不正确的原因是, 每个 case 结束后, 应该用 break 语句终止执行, 否则将顺序执行其后的每一个 case 及其 default 的语句。

【例题 12-9】 填空: 编写程序将在字符串 s 中出现的字符的正序和逆序连接, 形成一个新串在 t 中, t 中字符按原字符串中字符的顺序排列。例如, 当  $s="ABCDE"$  时, 其逆序为 "EDCBA", 结果应该是  $t="ABCDEEDCBA"$ 。

```
#include <conio.h>
```



```

#include <stdio.h>
#include <string.h>
void strcat_reverse (char *s, char *t)
{   int i, sl= strlen(s);
    for (i=0; i<sl; i++) t[i] = s[i];
    for (i=0; i<sl; i++) t[sl+i] = _____; /* 此处应填入 s[sl-i-1]; */
        /* 分析: 前一个 for 循环是复制原字符串 s 的正序, 本 for 循环显然是
            要复制原字符串 s 的逆序 */
    _____; /* 此处应填入 t[2*sl] = '\0' 或 t[2*sl] = 0 */
        /* 分析: 此处应该填入一个字符串的结束符 */
}
main( )
{   char s[100], t[100];
    printf("\nPlease enter string s:"); scanf("%s", s);
    strcat_reverse (s, t); printf("the result is: %s\n", t);
}

```

**【例题 12-10】** 填空: 在  $n$  行  $n$  列的矩阵中, 每行都有最大的数, 编写程序求这  $n$  个最大数中的最小一个。

```

# include <stdio.h>
# define N 100
int a[N][N];
void main( )
{   int row, col, max, min, n;
    scanf("%d", &n);
    for (row=0; row<n; row++)
        for (col=0; col<n; col++)
            scanf("%d", &a[row][col]);
    for (row=0; row<n; row++)
    {   for (max=a[row][0], col=1; col < n; col++)
        if (_____) max = a[row][col];
            /* 此处应该填入: a[row][col] > max */
        /* 分析: max 存放的是每行中的最大数, 这里是判断某个数是否大于 max,
            若是, 则 max = a[row][col]; 将这个数存为最大数 max */
        if (_____) min = max; /* 此处应该填入: row == 0 */
        /* 分析: min 存放的是 n 个最大数 max 中最小的一个, 但 min 没有赋过初
            值, min 肯定要取第一行 (row=0) 的最大值 max 作为初值 */
        else if (_____) min = max; /* 此处应该填入: max < min */
        /* 分析: min 存放的是 n 个最大数 max 中最小的一个, 这里是判断某个 max
            是否小于 min, 若是, 则 min=max 取这个 max 作为最小值 */
    }
}

```

```
    }  
    printf("The min of max numbers is %d\n", min);  
}
```

运行程序输入:

```
4  
1 2 3 4  
4 5 6 7  
7 8 9 10  
2 3 3 2
```

则输出结果为: The min of max numbers is 3

【例题 12-11】 对于程序

```
# include <stdio.h>  
# define N 4  
main( )  
{   int i, j, a[N][N];  
    for (i=0; i<N; i++)  
        for (j=0; j<N; j++)  
            scanf("%d", &a[i][j]); /* 从键盘读入二维数组 */  
    for (i=0; i<N; i++)  
    {   for (j=0; j<N; j++)  
        {   if (a[i][j] == 0)  
            continue;  
            if (a[i][j] < 0)  
                break;  
            printf(" %d", a[i][j]);  
        }  
        printf("\n");  
    }  
}
```

若从键盘上输入:

```
1 2 3 4  
5 0 6 9  
6 0 -1 2  
7 -2 8 6
```

请给出输出结果。

正确答案:

1 2 3 4  
5 6 9  
6  
7

分析：当元素值为 0 时，执行 `continue`，将不再执行后面的语句，而立即开始下一次循环，所以数值为 0 的数据不输出。当元素值小于 0 时，执行 `break`，将终止本层 `j` 的循环，本行数据不再输出。

**【例题 12-12】** 对于下面的程序，请给出输出结果。

```
# include <stdio.h>
# define N      10
# define s(x)    x*x
# define f(x)    (x*x)
# define g(x)    ((x)*(x))
main( )
{   int i1, i2, i3, i4;
    i1 = 1000/s(N);
    i2 = 1000/f(N);
    i3 = f(N+1);
    i4 = g(N+1);
    printf("i1=%d i2=%d i3=%d i4=%d\n", i1, i2, i3, i4);
}
```

正确答案：i1=1000 i2=10 i3=21 i4=121

分析：表达式 `1000/s(N)` 宏替换后为 `1000/10*10`，因此结果是 1000，而不是期待的 10。表达式 `1000/f(N)` 宏替换后为 `1000/(10*10)`，因此结果是 10，是期待的结果。表达式 `f(N+1)` 宏替换后为 `(10+1*10+1)`，因此结果是 21，而不是期待的 121。表达式 `g(N+1)` 宏替换后为 `((10+1)*(10+1))`，结果是 121，是期待的结果。

**【例题 12-13】** 对于下面的程序，请给出输出结果。

```
# include <stdio.h>
void f1(int a, int b, int c)
{   c = a + b; }
void f2(int *a, int *b, int *c)
{   *c = *a + *b; }
main( )
{   int a=3, b=4, c=5, d=6;
    f1(a, b, c);
    f2(&a, &b, &d);
}
```

```
printf("c=%d d=%d\n", c, d);
}
```

正确答案: c=5 d=7

分析: 对于函数 f1(a, b, c) 调用传参数时仅传值, 参数的值能传到函数但不能传回, 所以 c 的值仍为 5, 而不是期待的 7。而对于函数 f2(&a, &b, &d) 的调用, 传递的是地址, 所以 d 的值是期待的 7。

【例题 12-14】 对于下面的程序, 请给出输出结果。

```
#include <stdio.h>
void sort(int a[], int n)
{
    int i, j, t;
    for (i=0; i<n-1; i++)
        for (j=i+1; j<n; j++)
            if (a[i] < a[j]) { t=a[i]; a[i] = a[j]; a[j] = t; }
}
void main()
{
    int aa[10]={1,2,3,4,5,6,7,8,9,10}, i;
    sort(&aa[3], 5);
    for (i=0; i<10; i++) printf("%d, ", aa[i]);
    printf("\n");
}
```

正确答案: 1, 2, 3, 8, 7, 6, 5, 4, 9, 10,

分析: 只把从第 4 个元素开始的 5 个元素 4, 5, 6, 7, 8 进行了从大到小的排序, 而数组的其他元素未受影响。

【例题 12-15】 下面的程序中函数 sstrcmp() (功能等同于库函数 strcmp()) 是比较两个字符串是否相等, 当 s 串和 t 串相等时返回值为 0; 当 s 串大于 t 串时返回值为 >0; 当 s 串小于 t 串时返回值为 <0。请填空。

```
# include <stdio.h>
int sstrcmp(char *s, char *t)
{
    while (*s && *t && _____) /* 分析: *s && *t 是判断当 s 和 t 串都还未到串尾
                                     时, 等价于 *s !=NULL && *t !=NULL, 后面的空白
                                     处必然是要判断当前两个的字符是否相等, 所以应该填
                                     入: *s == *t */
    {
        s++; t++;
    }
    return _____; /* 分析: 这里填入的是函数的返回值, 而两个串之间的关系是由这两
                           个串第一个不同字符决定的, 所以应该填入: *s - *t */
}
```



```

void main( )
{   char str1[100], str2[100];
    scanf("%s%s", str1, str2); printf("%d\n", strcmp(str1, str2));
}

```

**【例题 12-16】** 请编写程序将字符串 s1 中出现的所有 s2 子串替换成 s3, 形成一个新串, 但不破坏字符串 s1。

参考答案是:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char *replace(char *s1, char *s2, char *s3)
{   char *p, *q, *r, *s; int l2, l3, l=0;
    l2 = strlen(s2);    /* 串 s2 的长度 */
    l3 = strlen(s3);    /* 串 s3 的长度 */
    p = s1;
    while ((p=strstr(p,s2)) != NULL)
    {   l++;            /* 统计 s2 串出现的次数 */
        p += l2;       /* 跳过本次出现的 s2 串 */
    }
    l = strlen(s1) - l*l2 + l*l3 + 1;    /* 计算替换后新字符串的长度 */
    s = r = (char *)malloc(l);           /* 分配动态内存存放新字符串 */
    p = s1;
    while (l)
    {   q = strstr(p, s2);    /* s2 是否在 s1 中出现, q 是首次出现的位置 */
        if (q != NULL)       /* q 不为空, 表示剩余的 s1 串中有 s2 */
        {   l = q - p;        /* l 为 s1 中出现 s2 串之前的长度 */
            strncpy(r, p, l); /* 将 s1 中出现 s2 串之前的子串复制到新串上 */
            r += l;           /* 将指向新串的指针移到串位 */
            strcpy(r, s3);    /* 将 s3 串复制到新串上, 即替换了 s1 中的 s2 子串 */
            r += l3;          /* 将指向新串的指针移到串位 */
            p = q + l2;        /* 将指向 s1 串的指针移到 s2 子串出现的位置后,
                               为下一次循环做好准备 */
        }
        else                  /* q 为空, 表示剩余的 s1 串中已经没有 s2 */
        {   strcpy(r, p);     /* 将 s1 串的剩余部分复制到新串的尾上 */
            break;            /* 终止循环 */
        }
    }
    return (s);              /* 返回指向所形成的新串的指针 */
}

```

```

}
void main()
{
    char *a="sabcababde", *b="ab", *c="efg", *d;
    d = replace(a, b, c);    printf("result=%s\n", d);    free(d);
}

```

运行结果: result=sefgcefgfgde

【例题 12-17】 下面的程序使用了函数指针, 请写出运行结果。

```

#include <stdio.h>
#include <math.h>
int f1(int x){ return x * x; }
int f2(int x) { return x * x * x; }
main()
{
    int x=3, y1, y2, y3, y4; int (*f)( );    /* 函数指针 */
    f = f1; y1 = (*f)(x);
    y2 = f1(x);
    f = f2; y3 = f(x);
    y4 = f2(x);
    printf("y1=%d y2=%d y3=%d y4=%d\n", y1, y2, y3, y4);
}

```

正确答案: y1=9 y2=9 y3=27 y4=27

分析:  $f = f1$  是将函数指针  $f$  指向了函数  $f1$ , 因此  $y1 = (*f)(x)$  是调用函数  $f1$ , 结果应该与  $y2 = f1(x)$  直接调用函数  $f1$  相同, 因此  $y1=y2=9$ 。同样  $f=f2$  是将函数指针  $f$  指向了函数  $f2$ ,  $y3 = f(x)$  是调用函数  $f2$ , 与直接调用函数  $f2$  等价, 因此  $y3=y4=27$ 。

【例题 12-18】 编写程序找出从自然数 1, 2, 3, ...,  $n$  中任取  $k$  个数的所有组合, 如  $n=5, k=3$ 。

分析: 这个问题可以用递归和非递归两种方法解决。

(1) 递归的方法

递归方法的关键是用归纳法找出一个递归数学公式, 这个组合问题可以这么想, 假如已经知道如何从  $n-1$  个数中取  $k$  个的方法, 并且有  $C_{n-1}^k$  种取法, 那么从  $n$  个中取  $k$  个, 可以归结为两部分:

- ① 从前  $n-1$  个数中取  $k$  个, 有  $C_{n-1}^k$  种取法;
- ② 先从前  $n-1$  个数中取  $k-1$  个, 再加上第  $n$  个数, 一共  $k$  个, 有  $C_{n-1}^{k-1}$  种取法。

取法总数:  $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$

可以看到两种趋势:

- $n$  逐步缩小, 直到与  $k$  相等, 就是从  $k$  个数中取  $k$  个, 取法只有一种, 就是这

k 个数全部;

- k 逐步缩小, 直到 k 等于 1, 就是从 n 个数中取 1 个, 取法有 n 种, 就是这 n 个数的每一个数构成的一个组合。

依据上述思路的递归程序是:

```
# include      <stdio.h>
# define      N    100
int a[N], nn;
void output(int a[], int nn)
{   int i; static int count=1;
    printf("%3d: ", count++);
    for (i=1; i<=nn; i++) printf("%d ", a[i]);
    printf("\n");
}
void comb(int n, int k)
{   if (n > k && k > 1)      /* 当 n 大于 k 并且 k 大于 1 时, 递归*/
    {   a[k] = n;           /* 先取 n 作为第一个元素 */
        comb(n-1, k-1);     /* 再在剩下的 n-1 个中取 k-1 个元素的组合*/
        comb(n-1, k);       /* 再在剩下的 n-1 个取 k 个元素的组合*/
    } /* 递归后, 把 n 个元素取 k 个的组合问题, 变成了 n-1 个中取 k 个、
        n-1 个中取 k-1 个的组合问题, 这样递归下去, 前一个会变成 n 等于 k 的组合问题,
        后一个会变成 k 等于 1 的问题 */
    else if (k == 1) /* 若 k 为 1, 则分别取其中每一个元素作为一个组合*/
    {   for (; n>0; n--) { a[k] = n;   output(a, nn); } }
    else if (n==k) /* 若 n 等于 k, 则将这 n 个元素作为一个组合*/
    {   for (; n>0; n--) a[n] = n;
        output(a, nn);
    }
}
void main( )
{   nn = 3; comb(5, nn); }
```

输出结果为:

1: 3 4 5

2: 2 4 5

3: 1 4 5

4: 2 3 5

5: 1 3 5

6: 1 2 5

7: 2 3 4

8: 1 3 4

9: 1 2 4

10: 1 2 3

## (2) 非递归的方法

虽然许多人对编写递归程序有些发怵,但反过来对于许多递归问题,转化成非递归的方法来编写程序,更是感觉困难。组合问题的非递归的解法方法多种多样,这里给出一种列举方法,希望能给读者一些启示。列举法实际上就是从  $n$  个数中先取出  $k$  个,从小到大有序地排列,比如:  $1, 2, \dots, k-1, k$  为一种组合,输出,然后从尾部开始将第  $k$  个数加 1,又是一种组合  $1, 2, \dots, k-1, k+1$ ,再输出, ..., 直到第  $k$  个数等于  $n$ ,组合为  $1, 2, \dots, k-1, n$ ,然后再将第  $k-1$  个数加 1,第  $k$  个数置为比第  $k-1$  个数大 1,组合为  $1, 2, \dots, k-2, k, k+1$ ,然后继续上面的过程,当第  $k-1$  个数到  $n-1$  时,组合为  $1, 2, \dots, k-2, n-1, n$ ,再将第  $k-2$  个数加 1,第  $k-1$  个数置为比第  $k-2$  个数大 1,第  $k$  个数置为比第  $k-1$  个数大 1,组合为  $1, 2, \dots, k-3, k-1, k, k+1$ ,再继续上面的过程,直到这  $k$  个数变成  $n-k+1, n-k+2, n-k+3, \dots, n-2, n-1, n$  为止。

依此思想的非递归程序是:

```
#include <stdio.h>
int comb[100];
void combine(int k, int n)
{
    int i, j; int flag; static int count = 1;
    do {
        flag = -1; printf("%3d: ", count);
        for(i = 0; i < k; i++) printf("%d ", comb[i]+1);
        printf("\n");
        for(i = k - 1; i >= 0; i--)
            if(comb[i] < n + i - k)
            {
                comb[i]++;
                for(j = i + 1; j < k; j++) comb[j] = comb[i] + j - i;
                flag = 0; count++; break;
            }
    } while (flag != -1);
}

void main( )
{
    int k=3, i;
    for(i = 0; i < k; i++) comb[i] = i;
    combine(k, 5);
}
```



**【例题 12-19】** 填空：下面程序各函数的主要功能是

**first\_insert()** 在已知链表的首表元之前插入一个指定值的表元；

**reverse\_copy()** 按已知链表复制出一个新链表，但新链表的表元链接顺序与已知链表的表元链接顺序相反；

**print\_link()** 用来输出链表中各表元的值；

**free\_link** 用来释放链表全部表元空间。

```
# include <stdio.h>
# include <malloc.h>
typedef struct node {
    int val;
    struct node *next;
} NODE;
void first_insert(NODE **p, int v)
{
    NODE *q = (NODE *)malloc(sizeof(NODE));
    q->val = v;
    _____; /* 分析：此函数的功能是在链表表首插入一个表元，前面的两个语句
                  分别是给表元 q 分配一个内存，并填入指定值 v，*p 是链表的头指
                  针，因此应该填入：q->next = *p，使 q 成为表首单元 */
    _____; /* 分析：表元 q 已经成为表首单元，链表的头指针*p 应该指向 q，因
                  此应该填入：*p = q */
}
NODE *reverse_copy(NODE *p)
{
    NODE *u;
    for (u = NULL; p; p=p->next) first_insert(_____);
    /* 分析：此函数是从已知链表 p 复制出一个新链表，表元链接顺序与已知链表 p 的表元
       链接顺序相反，显然 u 是指向新链表的表首指针。用 for 循环顺序从 p 链表中取得每
       一个表元值，用 first_insert() 插入到 u 链表表首，因此应该填入：&u, p->val
       */
    return u;
}
void print_link(NODE *p)
{
    for (; _____) printf("%d\t", p->val);
    /* 分析：此函数是用来输出链表中各表元的值，是从表首 p 开始，直到表尾，所以 for 的
       条件表达式 (第二个表达式) 应该是 p!=NULL 或直接填入 p；而 for 的第三个表达式应
       该是指针 p 指向下一个表元，应该是 p=p->next，因此此处应该填入：p; p=p->next
       或 p!=NULL; p=p->next */
    printf("\n");
}
void free_link(NODE *p)
```

```

{   NODE *u;
    while (p!= NULL)
    {   u = p->next;    free(p);
        _____; /* 分析: 此函数是用来释放链表全部表元空间, 是从表首 p 开始, 逐个
                        释放 u 是一个临时指针, 总是在释放当前表元 p 前, 记下它的下一个
                        表元, 因此在当前表元 p 释放后, 应该让指针 p 指向 u, 为下一次循
                        环做好准备, 因此此处应该填入: p = u */
    }
}

void main()
{   NODE *link1, *link2; int i;
    link1 = NULL;
    for (i=1; i<=10; i++) first_insert(&link1, i);
    link2 = reverse_copy(link1);
    print_link(link1); free_link(link1);
    print_link(link2); free_link(link2);
}

```

**【例题 12-20】** 对于下面的程序, 请给出运行结果。

```

#include <stdio.h>

void main()
{   FILE *fp; int i=20,a[100];float f=12.345f;
    fp = fopen("d1.dat", "w"); /* 以文本写方式打开文件 d1.dat */
    fprintf(fp, "%d %f\n", i, f); /* 向文件中写入一个整数和一个浮点数 */
    fclose(fp); /* 这时可以用任何文本编辑器打开文件 d1.dat 查看其内容, 为
                20 12.345000 */
    fp = fopen("d2.dat", "wb"); /* 以二进制写方式打开文件 d2.dat */
    for (i=0;i<100; i++) a[i] = i+100;
    fwrite(a, sizeof(int), 100, fp); /* 将 101 到 200 这 100 个二进制整数写
                                     到文件中 */
    fclose(fp); /* 文件 d2.dat 虽然已经存在, 但用文本编辑器打开看不到其内容 */
    fp = fopen("d1.dat", "r"); /* 以文本方式读打开文件 d1.dat */
    fscanf(fp, "%d%f", &i, &f); /* 从文件中分别读入一个整数和一个浮点数到
                                   i 和 f */
    printf("%d %f\n", i, f);    fclose(fp);
    fp = fopen("d2.dat", "rb"); /* 以二进制方式打开文件 d2.dat */
    fseek(fp, sizeof(int)*50L, SEEK_SET);
    /* 移动文件指针, 跳过前 50 个整数 */
    fread(a, sizeof(int), 50, fp); /* 读入 50 个二进制整数到 a 数组中, 这时 a
                                     数组中前 50 个数与后 50 个数是相同的, 都

```

是 151 到 200 \*/

```
fclose(fp);
}
```

**正确答案:** 20.12.345000

并在你的当前工作目录下生成了两个文件: dl.dat 和 d2.dat。

**【例题 12-21】** 填空: 下面的程序从合并文件中恢复出其中一个或全部原始文件。所有文件都以二进制方式处理。存放顺序是: 先逐个顺序存放各原始文件, 然后顺序存放各原始文件的控制信息: 文件名、文件长度和其在合并文件中的起始位置 (偏移量), 控制信息的格式如 FileInfo 结构, 在合并文件的最后存放一个特殊的 FileInfo 结构作为合并文件的结束标记: FileInfo EndFlag={"CombinedFile", 0, \_offset};

其中 \_offset 是第一个原始文件的控制信息在合并文件中的起始位置 (偏移量)。

本程序的运行格式是:

程序名 <合并文件名> [<原始文件名>]

这是用命令行参数的运行方式。如果不指定<原始文件名>, 此程序将默认恢复合并文件中的所有原始文件; 如果运行时没有输入命令行参数, 则程序会提示输入<合并文件名>。

```
# include <stdio.h>
# include <string.h>
typedef struct {
    char fname[256];
    long length;
    long offset;
} FileInfo;
void copyfile(FILE *fin, FILE *fout, int fsiz)
{ char buf[1024]; int siz=1024;
  while (fsiz != 0) /* 每次复制 siz 个字节, 直至复制完 fsiz 个字节 */
  { if (siz > fsiz) _____;
    /* 分析: 此函数是用来指定的输入文件指针 fin 中读入 fsiz 个字节, 写到
       指定的输出文件指针 fout 中, 按块处理, 每次 siz (1024) 个字节。如
       果要处理的文件长度 fsiz 不足 1024 时, 则按实际长度处理。因此此处应
       该填入: siz = fsiz */
    fread(buf, 1, siz, fin); fwrite(buf, 1, siz, fout);
    fsiz = _____; /* 分析: 每次复制完 siz 个字节后, 应该从总数 fsiz 中减去 siz, 因
       此此处应该填入: fsiz - siz */
  }
}
```

```
int dofile (FILE *fin, FileInfo *inp)
{ long offset; FILE *fout;
```

```

if ((fout = fopen(inp->fname, "wb")) == NULL)
{ printf("创建文件错误: %s\n", inp->fname); return 1; }
offset = ____; /* 保留合并文件读/写位置 */
/* 分析: 在开始读合并文件前, 先取得合并文件的当前读/写位置并用 offset
记录下以备恢复, 因此此处应该填入: ftell(fin) */
fseek(____); /* 定位于被恢复文件首 */
/* 分析: 要定位于被恢复文件首, 应该从文件头开始定位, 所以用 SEEK_SET
作为起始位置, 用被恢复文件的 inp->offset 作为位移。因此此处应该填
入: fin, inp->offset, SEEK_SET */
copyfile(fin, fout, inp->length); fclose(fout);
printf("\n---文件名: %s\n 文件长: %ld.\n", inp->fname, inp->length);
fseek(____); /* 恢复合并文件读/写位置 */
/* 分析: 要恢复合并文件读/写位置, 应该从文件头开始定位, 所以用 SEEK_SET 作
为起始位置, 用保存的合并文件读/写位置 offset 作为位移。因此此处应该填入:
fin, offset, SEEK_SET */
return 0;
}

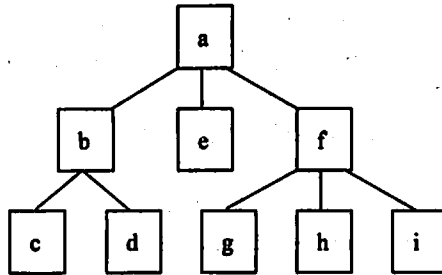
int main(int argc, char *argv[])
{ FileInfo finfo; char fname[256]; FILE *fcmbn;
  if (argc < 2) { printf("输入合并文件名:"); scanf("%s", fname); }
  else strcpy(fname, argv[1]);
  if ((fcmbn = fopen(fname, "rb")) == NULL)
  { printf("文件打开错误: %s\n", fname); return 1; }
  fseek(fcmbn, -sizeof(FileInfo), SEEK_END);
  /* 定位于合并文件末尾的标志信息 */
  fread(&finfo, 1, sizeof(FileInfo), fcmbn);
  if (finfo.length != 0 || strcmp(finfo.fname, "CombinedFile"))
  { printf("指定的文件不是合法的合并文件.\n"); fclose(fcmbn); return 2; }
  fseek(fcmbn, finfo.offset, SEEK_SET); /* 定位于第一个原始文件的控制信息 */
  for( ; ; ) /* 恢复一个 (argc > 2) 或全部 (argc = 2) 原始文件 */
  { fread(&finfo, 1, sizeof(FileInfo), fcmbn);
    if (finfo.length == 0) break;
    if (argc > 2 && strcmp(finfo.fname, argv[2])) continue;
    if (dofile(fcmbn, &finfo) != 0) break;
  }
  fclose(fcmbn); return 0;
}

```

**【例题 12-22】** 填空: 设 M 叉树采用列表法表示, 即每棵子树对应一个列表, 列表的结构为: 子树根结点的值后跟用 “( )” 括起来的各子树的列表 (若有子树的话), 各子树的列表间用 “,” 分隔。例如下面的三叉树可用列表 a(b(c,d),e,f(g,h,i)) 表示。



本程序根据输入的列表生产一棵 M 叉树，并由 M 叉树再输出列表。



```

#include <stdio.h>
#include <stdlib.h>
#define M 3          /* 三叉树 */
typedef struct node {
    int val;
    struct node *subTree[M];
} NODE;

char buf[255], *str=buf;
NODE *d = NULL;
NODE *makeTree()      /* 由列表生产 M 叉树 */
{
    int k; NODE *s;
    s = _____; /* 分析: 从本函数来看, s 是最终所生产的 M 叉树的根结点, 而 s 在前一个语句 NODE *s; 定义后, 并没有赋值, 所以本语句应该是为 s 分配动态内存, 因此此处应该填入: (NODE *)malloc(sizeof(NODE)) */

    s->val = *str++;
    for (k=0; k<M; k++)
        s->subTree[k] = NULL;
    if (*str == '(')
    {
        k=0;
        do {
            str++;
            s->subTree[k] = _____;
            /* 分析: 因为 s->subTree[k] 只能指向其子树, 所以本处应该递归调用 makeTree() 生成子树, 因此此处应该填入: makeTree() */
            if (*str == ')')
            {
                str++;
                break;
            }
            k=k+1;
        } while (_____);
    }
}

```

/\* 分析: 因为此函数是用 str 字符串来生成 M 叉树, 循环的终止条件必然是判断字符串是否到尾, 因此此处应该填入: \*str 或 \*str != NULL 或 \*str != '\0' 或 \*str != 0 \*/

```

    }
    return s;
}

void walkTree(NODE *t) /* 由 M 叉树输出列表 */
{
    int i;
    if (t != NULL)
    {
        _____; /* 分析: 若 M 叉树非空, 则首先开始输出 M 叉树的根结点的值, 由于根结点的值是字符型, 因此此处应该填入: putchar(t->val) 或 printf("%c", t->val) */
        if (t->subTree[0] == NULL)
            return;
        putchar('(');
        for (i=0; i<M; i++)
        {
            _____; /* 分析: 由于 M 叉树的当前子树非空, 应该递归调用 walkTree() 输出此子树, 因此此处应该填入: walkTree(t->subTree[i]) */
            if (i != M-1 && t->subTree[i+1] != NULL)
                putchar(',');
        }
        putchar(')');
    }
}

void main( )
{
    printf("Enter exp: ");
    scanf("%s", str);
    d = makeTree( );
    walkTree(d);
    putchar('\n');
}

```

**【例题 12-23】** 试给出下面程序的结果。

```

#include <iostream.h>
void swap(int &a, int &b) // 将 a, b 变量的值交换
{
    int t=a;
    a=b;
    b=t;
}

```

```

}
int main( )
{ int i=3, j=5;
  swap(i, j);      // i, j 的值被交换, i=5, j=3
  cout << i << j ; // 分析: 输出结果将是: 5 3
}

```

**【例题 12-24】** 试给出下面程序的结果。

```

#include <iostream.h>
void swap(int *a, int *b)    // 将 a, b 指针变量的值交换
{ int *t=a;
  a=b;
  b=t;
}
int main( )
{ int i=3, j=5;
  int *p=&i, *q=&j;          // 指针 p 指向 i, 而指针 q 指向 j
  swap(p, q);                // p, q 的值被交换, 指针 p 指向 j, 而指针 q 指向 i
  cout << *p << *q ;        // 分析: 输出结果将是: 5 3
}                             // 但 i, j 的值并未改变, 仍然是 i=3, j=5

```

**【例题 12-25】** 试给出下面程序中:: 的作用。

```

#include <iostream.h>
const int max=10000;
void main() // 从键盘上读入 10 个数, 求最大值 max 输出, 但最大不超过 10000
{ int max, i, j;
  cin >> max;
  for (j=1; j<10; j++)
  { cin >> i;
    if (i>max)
      max = i;
    if (max > ::max) // 若 max 大于外部变量 max 的值, 则取外部变量 max 的值
      max = ::max;
  }
  cout << max;
}

```

分析: ::使程序可以同时访问同名的局部变量和外部变量

**【例题 12-26】** 若有以下函数说明

```
void delay(int day, int month=1, int year=2003)
```

试说明函数调用 delay(24); delay(24, 5); delay(24, 5, 2002); 的参数结果。

分析: delay(24); 将使 day = 24, month = 1, year = 2003;

delay(24, 5); 将使 day = 24, month = 5, year = 2003;

delay(24, 5, 2002); 将使 day = 24, month = 5, year = 2002;

【例题 12-27】试给出下面程序的运行结果。

```
class MyClass {
public:
    MyClass(int i);
    ~MyClass( );
    void Print( );
private:
    int member;
};

MyClass::MyClass(int i)
{
    member = i;
    cout << "constructor called"<<member <<endl;
}

MyClass::~~MyClass( )
{
    cout << "destructor called"<<endl;
}

void MyClass::Print( )
{
    cout << "Print called =>" <<member <<endl;
}

void main( )
{
    MyClass Object(10); // Object 的初值为 10
    Object.Print( );
}
```

运行结果应该是:

constructor called 10

Print called => 10

destructor called

【例题 12-28】试说明下面程序中的 Add 函数为什么定义成友元。

```
class TRS
{
public:
    TRS(n, d);
    void Print( );
    friend TRS Add(TRS &r1, TRS &r2);
}
```



```

        private:
            long N, D;
    };
    ...
    TRS Add(TRS &r1, TRS &r2)
    { TRS r;
      r.N=r1.N*r2.D + r1.D*r2.N;
      r.D=r1.D*r2.D;
      return r;
    }
    int main()
    { TRS a(1,3), b(2,9), c;
      c = Add(a,b);
    }

```

分析: Add 函数不是成员函数, 并且要同时访问一个以上的对象的私有成员, 所以定义为 TRS 类的友元。

**【例题 12-29】** 试说明下面程序中求绝对值的函数 abs 为什么可以重名。

```

int abs(int i)
{ return i < 0 ? -i : i; }
long abs(long i)
{ return i < 0L ? -i : i; }
float abs(float i)
{ return i < 0.0 ? -i : i; }
void main( )
{ int i=-1; long L=-2L; float f=-1.5;
  i = abs(i); L=abs(L); f = abs(f);
}

```

分析: 程序中出现三个同名的 abs 函数, 分别对 int、long 和 float 类型数据求绝对值, 系统根据 i、L 和 f 的类型决定调用哪个 abs 函数。

**【例题 12-30】** 试给出下面程序的运行结果。

```

class pwr {
public:
    pwr(int x){num=x;}
    friend int operator ^(pwr, pwr);
private:
    int num;
};

```

```
int operator ^(pwr b, pwr e)
{ int t, temp= b.num;
  for (t=e.num-1;t;t--)
    temp *= b.num;
  return temp;
}
void main( )
{ pwr b(10), e(3);
  cout << b^e << 10^3;
}
```

分析：操作符 $\wedge$ 的原意是位与操作，我们进行操作符重载使其成为对 $b^e$ 求 $b^e$ 的值， $b^e$ 为1000。输出结果应该是：1000 9

## 12.3 思考练习题及答案

### 思考练习题

1. 填空：函数 `strcpy()` 将字符串 `from` 复制到 `to`，生成一个新串。

```
void strcpy(char *to char * from)
{ while (____(1)____);
}
```

2. 填空：函数 `merge` 将两个从小到大有序数组 `a` 和 `b` 合并生成一个新的从小到大有序整数序列 `c`，其中形参 `n` 和 `m` 分别是数组 `a` 和 `b` 的元素个数。

```
void merge(int a[ ], int n, int b[ ], int m, int *c)
{ int i, j;
  i=j=0;
  while (i<n && j<m)
    *c++ = a[i]<b[j]? a[i++]: b[j++];
  while (____(1)____)
    *c++ = a[i++];
  while (____(2)____)
    *c++ = b[j++];
}
```

3. 填空：递归函数 `sum(int a[], int n)` 的返回值是数组 `a` 的前 `n` 个元素之和。

```
int sum(int a[ ], int n)
{ if (n>0)
```

```

        return ____ (1) ____;
    else ____ (2) ____;
}

```

4. 填空: 函数 merge(intNode \*a, intNode \*b)将两个升序链表 a 和 b 合并成一个升序链表并返回首指针。

```

typedef struct element {
    int val;
    struct element *next;
} intNode;
intNode *merge(intNode *a, intNode *b)
{
    intNode *h=a, *p, *q;
    while (b)
    {
        for (p=h; p && p->val < b->val; q=p,p=p->next);
        if (p==h) ____ (1) ____;
        else ____ (2) ____;
        q=b; b=b->next;
        ____ (3) ____;
    }
    return h; /* 返回链表首指针 */
}

```

5. 填空: 递归函数 dec(int a[ ], int n)判断数组 a[ ]的前 n 个元素是否是不递增的。不递增返回 1, 否则返回 0。

```

int dec(int a[ ], int n)
{
    if (n <= 1) ____ (1) ____;
    if (a[0] < a[1]) return 0;
    return ____ (2) ____;
}

```

6. 填空: 设十进制长整数用数组存储, 如有 k 位的长整数 m 用数组 a[ ]存储:

$$m = a[k] \cdot 10^{k-1} + a[k-1] \cdot 10^{k-2} + \cdots + a[2] \cdot 10^1 + a[1] \cdot 10^0$$

并用 a[0]存储长整数 m 的位数, 即 a[0]=k。

存储长整数数组的每个元素只存储长整数 m 的一位数字, 长整数运算时, 为了运算方便, 产生的中间结果的某位数字可能会大于 9。这时, 就应该调用函数 format(int \*a)将它归整, 使数组的每个元素始终只存储长整数的一位数字。

```

void format( int *a)
{
    int p;

```

```

for (p=1; p<a[0] || a[p] >= 10; p++)
{
    if (p >= a[0]) (1);
    a[p+1] += a[p]/10;
    a[p] = (2);
}
if (p>a[0]) (3);
}

```

7. 填空: 函数  $\text{combine}(a,b,c)$  是计算两个整数的组合数。由于计算结果可能超出 long 整型的可表示范围, 故采用上题的数组方法存储计算结果。设整数  $a$  和  $b$  ( $a \geq b$ ), 它们的组合  $c(a,b)=a!/(a-b)!/b!$  可用以下方法计算:

$$c(a,b) = a*(a-1)*(a-2)*\cdots*(a-b+1)/b! = u_1 * u_2 * \cdots * u_b / (d_1 * d_2 * \cdots * d_b)$$

其中  $u_1=a, u_2=a-1, \dots, u_b=a-b+1; d_1=1, d_2=2, \dots, d_b=b$ 。

通过计算上述分式可以得到组合  $c(a,b)$ 。而为计算上述分式, 先从  $u_1, u_2, \dots, u_b$  中除掉  $d_1 * d_2 * \cdots * d_b$  的因子, 得到新的  $u_1, u_2, \dots, u_b$ 。然后再将它们相乘。以下函数中调用的外部函数  $\text{gcd}(a,b)$  是求两个整数  $a$  和  $b$  的最大公因子的函数, 函数  $\text{format}(a)$  是上题的函数。

```

#define MAXN    100
void combine(int a, int b, int *c)
{
    int i, j, x, k;
    int d[MAXN], u[MAXN];
    for (k=0, i=a; i >= a-b+1; i--) u[++k] = i;
    (1);
    for (i=1; i<=b; i++) d[i] = i; /* 将整数 1 至 b 顺序存于数组 d 中 */
    for (i=1; i<=u[0]; i++) /* 从 u 的各元素中, 去掉 d 中整数的所有因子 */
        if (u[i] != 1)
            for (j=1; j<=b; j++)
                if ((2))
                {
                    x=gcd(u[i], d[j]); u[i] /= x; d[j] /=x;
                }
    c[0] = c[1] = 1; /* 长整数 c 初始化 */
    for (i=1; i<=u[0]; i++) /* 将 u 中各整数相乘, 存于长整数 c 中 */
        if (u[i] != 1)
        {
            for (j=1; j<=c[0]; j++) c[j] = (3);
            format(c); /* 将存于 c 中的长整数规整 */
        }
}

```

8. 填空: 函数  $\text{encode}$  和  $\text{decode}$  分别实现对字符串的变换和复原。变换函数  $\text{encode}$



顺序考察已知字符串的字符, 按以下规则逐组生成新字符串:

(1) 若已知字符串的当前字符不是数字字符, 则赋值该字符到新字符串中。

(2) 若已知字符串的当前字符是一个数字字符, 且它之后没有后继字符, 则简单地将其复制到新字符串中。

(3) 若已知字符串的当前字符是一个数字字符, 并且还有后继字符, 设该数字字符的面值为  $n$ , 则将它后继字符(包括后继字符是一个数字字符)重复复制  $n+1$  次到新字符串中。

(4) 以上述一次变换为一组, 在不同组之间另插入一个下划线字符 '\_' 用于分隔。

例如: encode 函数对字符串 26a3t2 的变换结果为 666\_a\_tttt\_2。

复原函数 decode 做变换函数 encode 的相反工作。即复制不连续相同的单个字符, 而将一组连续相同的字符(不超过 10 个)变换成一个用于表示重复次数的数字字符和一个重复出现的字符, 并在复原过程中删除变换函数为不同组之间添加的一个下划线字符。

假定调用变换函数 encode 时的已知字符串中不包含下划线字符。

```
int encode(char *instr, char *outstr)
{
    char *ip, *op, c; int k, n;
    ip = instr; op = outstr;
    while (*ip)
    {
        if (*ip >= '0' && *ip <= '9' && *(ip+1) )
        {
            n = (1) ;
            c = (2) ;
            for (k=0; k<n; k++)
                *op++ = c;
        }
        else
            (3) ;
        *op++ = '_';
        ip++;
    }
    if (op > outstr)
        op--;
    (4) ;
    return op - outstr;
}

int decode(char *instr, char *outstr)
{
    char *ip, *op, c; int n;
    ip = instr; op = outstr;
    while (*ip)
    {
        c = *ip;
```



```

    n = 0;
    while (*ip == c && n < 10)
    {   ip++;
        n++;
    }
    if (____(5)____)
        *op++ = '0' + n - 1;
    *op++ = c;
    if (____(6)____)
        ip++;
}
*op = '\0';
return op - outstr;
}

void main( )
{   char p[ ]="26a3t2", q[20], r[20];
    encode(p, q);
    printf("p=%s q=%s\n", p, q);
    decode(q, r);
    printf("q=%s r=%s\n", q, r);
}

```

9. 填空：本程序从正文文件 text.in 中读入一篇英文短文，统计该短文中不同单词及出现次数，并按词典编辑顺序将单词及出现次数输出到正文文件 word.out 中。

程序用一棵有序二叉树存储这些单词及其出现的次数，边读入边建立，然后中序遍历该二叉树，将遍历经过的二叉树上的结点的内容输出。

```

#include <stdio.h>
#include <malloc.h>
#include <ctype.h>
#include <string.h>
#define INF "text.in"
#define OUTF "word.out"
typedef struct treenode {
    char *word;
    int count;
    struct treenode *left, *right;
} BNODE;
int getword(FILE *fpt, char *word)
{   char c;
    c=fgetc(fpt);

```

```

    if ( c == EOF)
        return 0;
    while(! (tolower(c) >= 'a' && tolower(c) <= 'z'))
    {
        c=fgetc(fpt);
        if ( c == EOF)
            return 0;
    } /* 跳过单词间的所有非字母字符 */
    while(tolower(c) >= 'a' && tolower(c) <= 'z')
    {
        *word++ = c;
        c = fgetc(fpt);
    }
    *word = '\0';
    return 1;
}

void binary_tree(BNODE **t, char *word)
{
    BNODE *ptr, *p; int compres;
    p = NULL; (1);
    while (ptr) /* 寻找插入位置 */
    {
        compres=strcmp(word, (2)); /* 保存当前比较结果 */
        if (!compres)
            (3); return;
        else
            (4);
        ptr = compres>0 ? ptr->right : ptr->left;
    }
    ptr = (BNODE *)malloc(sizeof(BNODE));
    ptr->left = ptr->right = NULL;
    ptr->word = (char *)malloc(strlen(word)+1);
    strcpy(ptr->word, word);
    ptr->count = 1;
    if (p == NULL)
        (5);
    else if (compres > 0)
        p->right = ptr;
    else
        p->left = ptr;
}

void midorder(FILE *fpt, BNODE *t)
{
    if ( (6) )
        return;

```



```

midorder(fpt, t->left);
fprintf(fpt, "%s %d\n", t->word, t->count);
midorder(fpt, t->right);
}

void main( )
{   FILE    *fpt; char word[40];
    BNODE    *root=NULL;
    if ((fpt=fopen(INF, "r")) == NULL)
    {   printf("Can't open file %s\n", INF);
        return;
    }
    while(getword(fpt, word) == 1)
        binary_tree( (7) );
    fclose(fpt);
    fpt = fopen(OUTF, "w");
    if (fpt == NULL)
    {   printf("Can't open file %s\n", OUTF);
        return;
    }
    midorder(fpt, root);
    fclose(fpt);
}

```

10. 填空: 本程序在  $3 \times 3$  方格中填入  $1 \sim N$  ( $N \geq 10$ ) 内的某 9 个互不相同的整数, 使所有相邻两个方格内的两个整数之和为质数。试求出满足这个要求的所有填法。 $3 \times 3$  方格中的每个方格按行按列(先行后列)序号为: 0, 1, 2, 3, 4, 5, 6, 7, 8。

程序采用试探法, 即从序号为 0 的方格开始, 为当前方格寻找一个合理的可填整数, 并在当前位置正确填入后, 为下一方格寻找可填入的合理整数。如不能为当前方格找到一个合理的可填整数, 就要回退到前一方格, 调整前一方格的填入整数。当直至序号为 8 的方格也填入合理的整数后, 就找到了一个解, 将该解输出。再调整序号为 8 的方格所填整数, 继续去找下一个解。为了检查当前方格的填入整数的合理性, 程序引入二维数组 `checkMatrix`, 存放需要进行合理性检查的相邻方格的序号。

```

#include <stdio.h>
#define N 12
int b[N+1];
int pos;
int a[9]; /* 用于存储诸方格所填入的整数 */
int AllNum=0; /* 统计有多少种填法 */
int checkMatrix[][3]={ {-1}, {0,-1}, {1,-1},

```



```

        {0,-1}, {1, 3,-1}, {2,4,-1}, {3,-1}, {4, 6,-1}, {5,7,-1} };
void write(int a[ ])
{   int i, j;
    for (i=0; i<3; i++)
    {   for (j=0; j<3; j++)
        printf("%3d", a[3*i+j]);
        printf("\n");
    }
}

int isPrime(int m)
{   int i;
    if (m==2) return 1;
    if (m==1 || m%2==0) return 0;
    for (i=3; i*i <= m; )
    {   if (m%i == 0) return 0;
        i += 2;
    }
    return 1;
}

int selectNum(int start)
{   int j;
    for (j=start; j<=N; j++)
        if (b[j]) return j;
    return 0;
}

int check() /* 检查填入 pos 位置的整数是否合理 */
{   int i, j;
    for (i=0; (j= (1) ) >= 0; i++)
        if (!isPrime(a[pos]+a[j]))
            (2);
    (3);
}

extend() /* 为下一方格找一个尚未使用过的整数 */
{   a[ (4) ] = selectNum(1);
    b[ a[pos] ] = 0;
}

void change() /* 为当前方格找下一个尚未使用过的整数。(找不到回溯) */
{   int j;
    while (pos >= 0 && (j=selectNum( (5) )) == 0)
        (6);
}

```

```

    if (pos<0) return;
    b[ a[pos] ] = 1; a[pos] = j; b[j] = 0;
}
int find( )
{
    int ok=1;
    pos=0; a[pos]=1; b[ a[pos] ] = 0;
    do {
        if (ok)
            if ( (7) )
            {
                write(a);
                change( );
                AllNum++; /* 统计有多少种填法 */
            }
            else extend( );
        else change( );
        ok = check( );
    } while (pos >=0);
}
void main( )
{
    int i;
    for (i=1; i<=N; i++) b[i] = 1;
    find( );
    printf("共有%d 种不同填法!\n", AllNum);
}

```

11. 填空：以下 C++程序的功能是计算三角形、矩形和正方形的面积并输出。

程序由 4 个类组成：类 Triangle、Rectangle 和 Square 分别表示三角形、矩形和正方形；抽象类 Finure 提供了一个纯虚拟函数 getArea ( )，作为计算上述三种图形面积的通用接口。

```

#include<iostream.h>
#include<math.h>
class Figure{
public:
    virtual double getArea( )=0;    //纯虚拟函数
};
class Rectangle: (1) {
protected:
    double height;
    double width;
public:

```

```

    Rectangle( ) { };
    Rectangle(double height,double width){
        This->height=height;
        This->width=width;
    }
    double getarea( ){
        return (2) ;
    }
};
class Square: (3) {
public:
    square(double width){
        (4) ;
    }
};
class triangle: (5) {
    double la;
    double lb;
    double lc;
public:
    triangle(double la ,double lb,double lc){
        this ->la=la; this->lb; this->lc;
    }
    double getArea(){
        double s=(la +lb+lc)/2.0;
        return sqrt(s*(s-la)**(s-lb) *(s-lc));
    }
};
void main( ){
    figure* figures[3]={
        new triangle(2,3,3),new Rectangle(5,8),new Square(5));
    for (int i=0;i<3;i++){
        cout<<"figures["<<i<<" ]area= "<<(figures[i])->getarea()<<endl;
    }
}

```

## 思考练习题答案

1. (1) \*to++ = \*from++

分析：本函数是进行字符串复制，而本填空是 while 循环的条件表达式，循环体为空，对应字符赋值、指针各自加 1、判断是否到字符串尾这几步操作都必须在这个填空

中完成, 因此应该填入: `*to++ = *from++`。

2. (1) `i < n` 或 `i <= n-1`

(2) `j < m` 或 `j <= m-1`

分析: 本函数是进行有序数组的合并, 在填空前面的 `for` 循环结束后, 两个数组必然有一个还没有到达尾部 (`i < n` 或 `j < m`), 因此需要继续用循环语句复制, 直到指针到数组尾。因此应该填入: `i < n` 和 `j < m`。

3. (1) `sum(a, n-1) + a[n-1]` 或 `a[n-1] + sum(a, n-1)`

分析: 本函数是递归求数组 `n` 个元素的和, 方法必然是先递归求出前 `n-1` 个元素的和, 再加上第 `n` 个元素。因此应该填入: `sum(a, n-1) + a[n-1]`。

(2) `return 0` 或 `return(0)`

分析: 此填空位于 `else` 语句中, 即 `n=0` 没有元素, 返回值为 0。因此应该填入: `return 0`。

4. (1) `h=b`

(2) `q->next=b`

分析: 本函数是进行有序链表的合并, 方法是循环将 `b` 链表的每一个结点插入到 `a` 链表中, 本处填空前面的 `for` 循环是为 `b` 结点找合适的插入位置 `q` (插入其后), 当循环结束后, 若 `p==h`, 则 `b` 结点将插入在表头的位置, 首指针 `h` 应该指向 `b`, 因此(1)应该填入: `h=b`。而当 `p!=h` 时, `b` 应该插入在 `q` 之后, 因此(2)应该填入: `q->next=b`。

(3) `q->next=p`

分析: `b` 要插入到 `q` 和 `p` 之间, 前面的语句已经将 `b` 链接在 `q` 之后, 还需要将 `p` 链接在 `b` 结点之后, 由于前面有语句已经将 `q` 指向 `b`, 因此应该填入: `q->next=p`。

5. (1) `return 1`

分析: 本函数判断 `a` 数组的前 `n` 个元素是否不递增, 若 `n` 小于等于 1, 则必然是不递增的, 所以返回值为 1。因此应该填入: `return 1`。

(2) `dec(a+1, n-1)` 或 `dec(&a[1], n-1)`

分析: 当 `n` 大于 1 时, 判断前两个元素 `a[0]` 和 `a[1]` 是否是不递增的, 是递增则返回 0 结束; 不是则需要递归调用本函数判断从 `a[1]` 开始的 `n-1` 个元素是否是不递增的。因此应该填入: `dec(a+1, n-1)` 或 `dec(&a[1], n-1)`。

6. (1) `a[p+1] = 0`

分析: 由于本处填空是在 `if(p >= a[0])` 后, 而 `for` 循环在 `p >= a[0]` 时能够执行到此语句, 必然是 `a[p] >= 10`, 即 `a[p]` 需要规整, 规整 `a[p]` 必然要产生新的进位 `a[p+1]`, 由于 `p` 是目前数组的最高位, `a[p+1]` 在使用前需要初始化为 0。因此应该填入: `a[p+1] = 0`。

(2) `a[p] % 10`

分析: 本处填空是规整第 `p` 位的整数, 由于前一个语句: `a[p+1] += a[p]/10` 将 `a[p]` 大于等于 10 的部分进位到了 `p+1` 位上, 因此 `a[p]` 只需保留小于 10 的部分。因此应该填



入:  $a[p] \% 10$ 。

(3)  $a[0] = p$

分析: 本处填空在条件语句  $\text{if}(p > a[0])$  后, 若  $p > a[0]$  成立, 则说明整数规整后, 位数已经突破了原来的  $a[0]$  位,  $a[0]$  应该记录当前的  $p$  位。因此应该填入:  $a[0] = p$ 。

7. (1)  $u[0] = k$

分析: 本处填空在  $\text{for}$  语句后,  $\text{for}$  语句的赋值语句  $u[++k]=i$  是给  $u[1], u[2], \dots$  赋初值,  $k$  为  $u$  的位数, 按照第 6 题存长整数的约定,  $u[0]$  应该存这个整数的位数。因此应该填入:  $u[0] = k$ 。

(2)  $d[j] > 1$

分析: 本处填空是一个  $\text{if}$  语句的条件表达式, 是循环从  $u$  数组中去掉所有  $d$  数组的因子, 当  $d[j]$  大于 1 时就需要去找最大公因子。因此应该填入:  $d[j] > 1$ 。

(3)  $c[j] * u[i]$

分析: 本处填空在一个两重循环中, 是将  $u$  中各整数相乘存入  $c$  中, 需要将  $c$  的每一位与  $u$  的每一位相乘。因此应该填入:  $c[j] * u[i]$ 。

8. (1)  $*ip - '0' + 1$

分析: 本处填空将数字字符转化成相应数字加 1 (即  $n+1$ )。因此应该填入:  $*ip - '0' + 1$ 。

(2)  $*++ip$

分析: 本处填空是取数字字符的后继字符。因此应该填入:  $*++ip$ 。

(3)  $*op++ = *ip$

分析: 本处填空是当字符  $*ip$  不是数字字符时, 复制该字符到新字符串中。因此应该填入:  $*op++ = *ip$ 。

(4)  $*op = '\0'$  或  $*op = 0$

分析: 本处填空是给新字符串赋一个串结束符号。因此应该填入:  $*op = '\0'$  或  $*op = 0$ 。

(5)  $n > 1$  或  $n \geq 2$

分析: 本处填空的前面是在统计是否有连续相同的字符, 因此此处应该判断连续相同的字符是否多于 1 个。因此应该填入:  $n > 1$  或  $n \geq 2$ 。

(6)  $*ip == '_'$

分析: 本处填空判断当前字符是否为下划线, 是则跳过。因此应该填入:  $*ip == '_'$ 。

9. (1)  $ptr = *t$

分析: 本处填空是函数 `binary_tree` 的开始处, 进行初始化, 应该是让指针  $ptr$  指向树的根结点  $*t$ 。因此应该填入:  $ptr = *t$ 。

(2)  $ptr \rightarrow \text{word}$  或  $(*ptr).\text{word}$  或  $ptr[0].\text{word}$

分析: 本处填空将要插入的单词  $\text{word}$  与当前指针  $ptr$  所指的结点的  $\text{word}$  比较大小。

因此应该填入: `ptr->word`。

(3) `ptr->count++`

分析: 本处填空是当要插入的单词 `word` 与指针 `ptr` 所指的结点的 `word` 相同时的处理, 必然是将指针 `ptr` 所指结点的计数器 `count` 加 1。因此应该填入: `ptr->count++`。

(4) `p=ptr`

分析: 本处填空是当要插入的单词 `word` 与指针 `ptr` 所指结点的 `word` 不相同时的处理, 必然是让 `p` 指向 `ptr`, 而 `ptr` 指向其左子树或右子树。因此应该填入: `p=ptr`。

(5) `*t = ptr`

分析: 本处填空是当 `p` 为空时的处理, 应该是让树的根结点指针指向 `ptr` 以便返回。因此应该填入: `*t = ptr`。

(6) `t=NULL` 或 `t=0`

分析: 本处填空是 `if` 语句的条件表达式, 若为真则不做任何事情立即返回, 只有当树为空才会出现这种情况。因此应该填入: `t=NULL`。

(7) `&root, word`

分析: 本处填空是在循环读入单词并插入树中, 为了能返回插入后的树根结点指针, 需要将 `root` 的地址传给 `binary_tree` 函数。因此应该填入: `&root, word`。

10. (1) `checkMatrix[pos][i]`

分析: 本处填空是在循环检查填入 `pos` 位置的整数是否合理, 把与 `pos` 相邻的数都求和判断是否为质数。因此应该填入: `checkMatrix[pos][i]`。

(2) `return 0`

分析: 若不是质数则返回 0, 表示不可以。因此应该填入: `return 0`。

(3) `return 1`

分析: 若相邻的数都是质数则返回 1, 表示可以。因此应该填入: `return 1`。

(4) `++pos`

分析: 本处填空是为下一个方格找一个尚未使用过的整数。因此应该填入: `++pos`。

(5) `a[pos]+1`

分析: 本处填空是在循环为当前方格找下一个尚未使用过的整数。因此应该填入: `a[pos]+1`。

(6) `b[ a[pos--] ] = 1`

分析: 本处填空是回溯找一个尚未使用过的整数。因此应该填入: `b[ a[pos--] ] = 1`。

(7) `pos == 8`

分析: 本处填空是判断当序号是否为 8, 是则说明序号为 8 的方格也填入了合理的整数, 输出这个解。因此应该填入: `pos == 8`。

此程序运行结果为: .....共有 768 种不同填法!

11. (1) `public Figure`

分析：本处由于 Rectangle 是派生类，需要公有继承 Figure。因此应该填入：public Figure。

(2) height\*width 或 width\*height 或 this->height\* this->width  
或 this->width\* this->height

分析：本处是计算矩形的面积。因此应该填入：height\*width。

(3) public Rectangle

分析：本处由于 Square 是派生类，需要公有继承 Rectangle。因此应该填入：  
public Rectangle。

(4) this->height = this->width = width 或 height = this->width = width

分析：本处是正方形的构造函数，是给继承的 Rectangle 赋初值，由于正方形长等于宽，因此应该填入：this->height = this->width = width 或 height = this->width = width。

(5) public Figure

分析：本处由于 Triangle 是派生类，需要公有继承 Figure。因此应该填入：public Figure。

## 1998-2007 年全国计算机技术与软件专业资格(水平)考试真题及答案汇总

2007 年下半年软考网络工程师试题 Word版  
2007 年下半年信息系统项目管理师试题 Word版  
2007 年下半年信息系统监理师试题 Word版  
2007 年下半年信息处理技术员试题 Word版  
2007 年下半年系统分析师试题 Word版  
2007 年下半年网络管理员试题 Word版  
2007 年下半年数据库系统工程师试题 Word版  
2007 年下半年软件设计师试题 Word版  
2007 年下半年嵌入式系统设计师试题 Word版  
2007 年下半年多媒体应用设计师试题 Word版  
2007 年下半年电子商务设计师试题 Word版  
2007 年下半年电子商务技术员试题 Word版  
2007 年下半年程序员试题 Word版  
2007 年上半年系统分析师试题 PDF版  
2007 年上半年信息系统监理师试题 PDF版  
2007 年上半年信息系统管理工程师试题 PDF版  
2007 年上半年信息处理技术员试题 PDF版  
2007 年上半年网络管理员试题 PDF版  
2007 年上半年数据库系统工程师试题 PDF版  
2007 年上半年网络工程师试题 PDF版  
2007 年上半年网络工程师试题答案 Doc  
2007 年上半年软件设计师试题 PDF版  
2007 年上半年软件设计师试题答案 Doc  
2007 年上半年软件评测师试题 PDF版  
2007 年上半年程序员题 PDF版  
2006 年下半年信息系统项目管理师试题 PDF版  
2006 年下半年系统分析师试题 PDF版  
2006 年下半年信息系统监理师试题 PDF版  
2006 年下半年网络工程师试题 PDF版  
2006 年下半年网络工程师试题及答案 Word版  
2006 年下半年软件设计师试题 PDF版  
2006 年下半年软件设计师试卷及答案 Word版  
2006 年下半年嵌入式系统设计师试题 PDF版



2006 年下半年电子商务设计师试题 PDF版  
2006 年下半年网络管理员试题 PDF版  
2006 年下半年程序员试题 PDF版  
2006 年上半年程序员试题 Word版  
2006 年上半年网络工程师试题(含答案) Word版  
2006 年上半年软件设计师试题及答案 Word版  
2005 年下半年软件设计师试题及答案 Word版  
2005 年下半年网络工程师试题及答案 Word版  
2005 年下半年网络管理员试题  
2005 年上半年网络工程师试题(含答案) Word版  
2005 年上半年软件设计师试题(含答案) Word版  
2005 年上半年程序员上午试题(word版)  
2005 年上半年网络管理员试题  
2005 年上半年网络管理员上午试题(word版)  
2005 年上半年网络管理员下午试题(word版)  
2005 年上半年网络工程师上午试题(word版)  
2005 年上半年网络工程师下午试题(word版)  
2005 年上半年软件设计师上午试题(word版)  
2005 年上半年软件设计师下午试题(word版)  
2004 年下半年网络工程师试题(含答案) Doc  
2004 年下半年软件设计师试题(含答案) Word版  
2004 年下半年网络工程师上午试题(word版)  
2004 年下半年网络工程师下午试题(word版)  
2004 年下半年程序员上午试题(word版)  
2004 年上半年网络工程师试题(含答案) Word版  
2004 年上半年软件设计师试题含答案 Word版  
2004 年下半年网络管理员员试题  
2004 年上半年软件设计师(高级程序员)上午试题(word版)  
2004 年上半年软件设计师(高级程序员)下午试题(word版)  
2004 年上半年程序员上午试题(word版)  
2004 年上半年程序员下午试题(word版)  
2003 年高级程序员试题(含答案) Word版  
2003 年度网络设计师试题及答案 Word版  
2003 年网络设计师上(下)午试题(word版)  
2003 年程序员考试上(下)午试题(word版)

2003 年初级程序员上午试题(word版)  
2003 年初级程序员下午试题(word版)  
2002 年度网络设计师级试题及答案 Word版  
2002 年度高级程序员级试题答案  
2002 年网络程序员试题  
2002 年系统设计师(高级程序员)上午试题(word版)  
2002 年系统设计师(高级程序员)下午试题(word版)  
2002 年系统分析员级上(下)午试题(word版)  
2002 年网络程序员级上(下)午试题(word版)  
2001 年度网络设计师级试题及答案 Word版  
2001 年度高级程序员级试题及答案 Word版  
2001 年高级程序员级下午试题(word版)  
2001 年高级程序员级上午试题(word版)  
2001 年网络设计师考试上午试题(word版)  
2001 年网络设计师考试上午试题(word版)  
2001 年程序员上午试题(word版)  
2001 年初级程序员级下午试题(word版)  
2001 年度网络程序员级试题  
2001 年系统分析员级上午试题(word版)  
2000 年高级程序员级试题及答案 Word版  
2000 年初级程序员级上午试题(word版)  
2000 年初级程序员级下午试题(word版)  
2000 年初高程序员级上午试题(word版)  
2000 年初高程序员级下午试题(word版)  
2000 年程序员上午试题(word版)  
1999 年初级程序员上午试题(word版)  
1999 年初级程序员级下午试题(word版)  
1999 年程序员上午试题(word版)  
1999 年程序员下午试题(word版)  
1999 年高级程序员上午试题及答案(word版)  
1999 年高级程序员下午试题(word版)  
1998 年高级程序员级上午试题(word版)  
1998 年高级程序员级下午试题(word版)

## 计算机技术与软件专业资格考试推荐视频教程下载汇总

数据结构视频教程 清华大学严蔚敏主讲 全 48 讲 ASF格式

数据结构C语言版视频教程 全 52 讲

操作系统原理视频教程 浙江大学徐宗元教授主讲(全 32 讲)

数据结构与算法 浙大徐镜春教授主讲(全 40 讲)

计算机网络基础 陆魁军主讲(全 32 讲)

数据库系统概论 浙大张军主讲(全 32 讲)

软件工程基础 浙大陈天洲主讲(全 32 讲)

面向对象程序设计 浙大毛根生主讲(全 30 讲)

### 程序设计语言视频教程:

孙鑫C++视频教程 rmvb格式 全 20CD

中山大学蔡培兴 C++语言视频教程 全 51 讲

Java视频教程 孙鑫Java无难事(全 12CD)

Java视频教程 即学即会java

中山大学汇编语言视频教程 全 51 讲

汇编语言程序设计视频教程(郝玉洁主讲 全 36 讲)

### 其他专项视频教程系列:

[程序员考试视频教程] 硬件基础 二进制编码

[程序员考试视频教程] 图的遍历

[程序员考试视频教程] 数据库 关系运算

[程序员考试视频教程] 操作系统 作业调度算法

[网络工程师培训视频教程] 帧中继拥塞控制

[网络工程师培训视频教程] 以太网CSMA/CD

[网络工程师培训视频教程] 奈氏准则和香农公式

[网络工程师培训视频教程] 滑动窗口模型

[网络工程师培训视频教程] TCP数据传输

[信息系统项目管理师视频教程]前言

[信息系统项目管理师视频教程]论文写作 注意事项及例题

[信息系统项目管理师视频教程]项目管理案例分析 例题讲解

[信息系统项目管理师视频教程]项目组合管理 盈亏平衡点

[信息系统项目管理师视频教程]项目质量管理 软件质量特性

[信息系统项目管理师视频教程]项目进度管理 关键路径

[信息系统项目管理师视频教程]项目沟通管理 项目管理信息系统



[信息系统项目管理师视频教程]项目风险分析 决策树分析  
[信息系统项目管理师视频教程]项目范围管理 实现范围变更  
[信息系统项目管理师视频教程]项目成本管理 挣值法的例子  
[信息系统项目管理师视频教程]项目成本管理 影响成本估算的因素  
[信息系统项目管理师视频教程]成本效益分析 动态投资回收期  
[软件设计师视频教程] 最小生成树  
[软件设计师视频教程] 知识产权-著作权法解读  
[软件设计师视频教程]数据流图设计(例题讲解)  
[软件设计师视频教程] 数据安全-对称加密  
[软件设计师视频教程] 面向对象继承访问控制  
[软件设计师视频教程] 操作系统习题讲解  
[软件设计师视频教程] **Web Service**及应用



## **.NET 语言(C#/VB)电子资料、开发工具下载汇总**

### **Visual Studio 相关电子资料、软件汇总:**

VS代码辅助工具Visual Assist X 10.4 完美版+特别文件  
CodeSmith 4.1.2 专业版 最新完美版 .NET代码模板生成工具  
Altova MissionKit 2008 for Enterprise Software Architects完美版  
正则表达式辅助生成工具RegexBuddy 3.0.5 破解版  
Pro Visual Studio 2005 Team System  
Microsoft Visual Studio 2005 Unleashed  
Visual Studio Team System Better Software Development for Agile Team

### **.NET 开发语言电子资料汇总:**

Pro C# 2008 and the .NET 3.5 Platform  
Apress出版 Accelerated C# 2008  
Pro LINQ:Language Integrated Query in C# 2008  
MS Press - Introducing Microsoft LINQ  
LINQ for Visual C# 2005 (07 年 6 月出版)  
LINQ for VB 2005 (07 年 6 月最新PDF文字版)  
Wrox C# 入门经典  
C# 设计模式  
C# 网络核心编程  
Windows应用高级编程 C#编程篇  
C#高级编程(第三版)  
数据结构与算法 C#语言版  
C#字符串和正则表达式参考手册  
O'Reilly 正则表达式参考手册 第二版 2007 年最新出版  
Programming Microsoft Windows with C#  
C# 2005 图解教程  
Visual C# 2005 Express Edition编程初学者指南  
Programming .NET Framework with C#  
C#语言参考  
C#应用程序开发  
Client Side Reporting with Visual Studio in C#  
Wrox Professional VB 2005 with .NET 3.0

Build A Program Now Visual Basic 2005  
.NET游戏编程入门经典—VB.NET篇  
O'Reilly Visual Basic 2005 Cookbook  
.NET Insight for Classic VB Developers  
Fast Track Visual Basic.NET  
How to Code .NET  
ADO.NET全攻略  
Apress Professional ADO.NET 2.0  
Oreilly .NET and XML  
.NET组件编程 (第二版)  
Wrox Beginning Visual C++ 2005  
Visual C++.NET专业项目  
精通.Net核心技术原理与构架  
Cross-Platform Web Services Using C# and Java  
Advanced C# Programming

## **.NET、ASP.NET 控件及源码大汇总**

[CuteEditor 6.0 在线HTML编辑器的领航者](#)

[ComponentArt.Charting.WebChart.dll](#)

[ComponentArt.Web.UI 2007.2 源代码+实例+DLL](#)

[ComponentArt.WebUI.2007.1 破解DLL](#)

[ComponentArt.WebUI.2007.1 源代码](#)

[ComponentArt.Web.UI.2006.2](#)

[ComponentArt.Web.UI.2006.2 源代码](#)

[ComponentArt.Web.UI.2006.1](#)

[Infragistics NetAdvantage for ASP.NET 2007 Vol 2](#)

[Infragistics NetAdvantage AppStylist 2007 Vol 2](#)

[Infragistics TestAdvantage WinForms 2007 For CLR2](#)

[Infragistics TestAdvantage WinForms 2007 for CLR1.x](#)

[Infragistics NetAdvantage for Windows Forms 2007 Vol 2](#)

[Infragistics NetAdvantage 2007 for WPF](#)

[Infragistics NetAdvantage 2006 Vol2 CLR1.x](#)

[Infragistics NetAdvantage 2006 Vol2 for CLR2](#)

[Infragistics NetAdvantage 2006 Vol2 CLR1.x](#)

[ComponentOne Studio 2007 v1.5 for ASP.NET 2.0](#)

[ComponentOne Studio 2007 v1.5 for ASP.NET 1.x](#)

[ComponentOne Studio 2006 v2 for ASP.NET 2.0](#)

[ComponentOne Studio 2006 v2 for ASP.NET 1.x](#)

[ComponentOne Studio for Mobile Devices 2007 v1.5 CLR1x](#)

[ComponentOne Studio for Mobile Devices 2006 v2 CLR2](#)

[ComponentOne Studio for Mobile Devices 2006 v2 CLR1.x](#)

[ComponentOne Studio 2007 v1.5 for .NET CLR2](#)

[ComponentOne Studio 2007 v1.5 for .NET CLR1.x](#)

[ComponentOne Studio for .NET 2006 v2 CLR2](#)

[ComponentOne Studio for .NET 2006 v2 CLR1.x](#)

[ComponentOne Studio for ActiveX 2007 v1.5](#)

[ComponentOne Studio for ActiveX 2006 v2 CLR2](#)

[ComponentOne Studio for ActiveX 2006 v2 CLR1.x](#)

[Telerik RadWindow for ASP.NET 2.0 v1.8.2.0](#)

[Telerik RadUpload for ASP.NET 2.0 v2.3.2.0](#)

Telerik RadTreeView for ASP.NET 2.0 v6.2.2.0

Telerik RadTabStrip for ASP.NET 2.0 v3.5.2.0

Telerik RadToolBar for ASP.NET 2.0 v1.5.2.0

Telerik RadSplitter for ASP.NET 2.0 v1.2.2.1

Telerik RadSpell for ASP.NET 2.0 v3.1.2.0

Telerik RadRotator for ASP.NET 2.0 v2.6.2.0

Telerik RadPanelbar for ASP.NET 2.0 v4.2.2.0

Telerik RadMenu for ASP.NET 2.0 v4.2.2.0

Telerik RadInput for ASP.NET 2.0 v2.0.2.0

Telerik RadGrid for ASP.NET 2.0 v4.6.2.0

Telerik RadEditor for ASP.NET 2.0 v7.1.2.0

Telerik RadComboBox for ASP.NET 2.0 v2.7.2.0

Telerik RadDock for ASP.NET 2.0 v1.8.2.0

Telerik RadChart for ASP.NET 2.0 v3.2.1.0

Telerik RadCalendar for ASP.NET 2.0 v2.1.2.0

Telerik RadAjax for ASP.NET 2.0 v1.7.2.0

telerik r.a.d.upload

telerik r.a.d.window

telerik r.a.d ToolBar

telerik r.a.d.Chart

telerik r.a.d.combobox

DotNetBar for VS2005 6.8.0.1

DotnetCharting 4.3 破解DLL

DotNET Charting WebForms

dotnetCharting.WinForms

TeeChart for .NET 3.2.2763.26084 完美DLL

TeeChart for .NET 3.2.2699.17379 完美DLL

DevExpress 7.3.4 完美破解DLL

Dxperience 7.3.5 完美破解DLL

DevExpress.LocalizationCHS.Dll

NickLee.Web.UI

SolpartWebControls

AspNetPager 6.0 for ASP.NET 1.x 自定义分页控件

AspNetPager 6.0 for ASP.NET 2.0 自定义分页控件

数据操作类 Socut.Data.dll for .NET 2.0 v3.1



数据操作类 Socut.Data.dll for .NET 1.x v3.1

Developer Express for .NET v7.3.5.0 全套完美无限制版

## **ASP.NET、Ajax、Silverlight 学习电子资料汇总**

### **Silverlight 电子资料汇总:**

[O'Reilly Silverlight 1.1 简介](#)

[Wrox出版 Silverlight 1.0 \(彩页染色代码、全面解析\)](#)

[Silverlight 1.0 Development with JavaScript](#)

[Sams出版 Silverlight 1.0 Unleashed](#)

[O'Reilly Essential Silverlight](#)

[XAML简明教程 CHM+PDF版](#)

### **ASP.NET 1.x/2.0/3.5 电子资料汇总:**

[Pro ASP.NET 3.5 in C# 2008](#)

[Beginning ASP.NET 3.5 in VB 2008 从入门到精通](#)

[Wrox ASP.NET 2.0 MVP Hacks and Tips](#)

[Professional ASP.NET.2.0 Design](#)

[ASP.NET2.0 入门经典](#)

[Wrox ASP.NET 2.0 Visual Web Developer 2005 Express Edition Starter](#)

[Beginning ASP.NET 2.0 in C# 2005 From Novice to Professional](#)

[Wrox Professional ASP.NET 2.0](#)

[Wrox Professional ASP.NET 2.0 XML](#)

[Wrox Professional ASP.NET 2.0 Security Membership and Role Management](#)

[Wrox Beginning ASP.NET 2.0 and Databases](#)

[ASP.NET开发人员手册](#)

[ASP.NET 2.0 网络编程入门到精通](#)

[ASP.NET Web应用程序开发新思维](#)

[ASP.NET 2.0 高级应用程序设计专家教程](#)

[ASP.NET XML高级编程 C#编程篇](#)

[ASP.NET程序开发 C#篇](#)

[ASP.NET XML深入编程技术](#)

[ASP.NET 2.0 Cookbook](#)

[ASP.NET 2.0 Everyday Apps for Dummies](#)

[Pro ASP.NET for SQL Server](#)

[ASP.NET 从入门到精通](#)

[Wrox Beginning ASP.NET 1.1 with Visual C#.NET 2003](#)

## ASP.NET 2.0 揭秘

Build Your Own ASP.NET 2.0 Web Site Using C# and VB

开发Microsoft ASP.NET 2.0 网络应用程序

开发ASP.NET 2.0 核心参考

Building Websites with VB.NET and DotNetNuke 4

Wrox出版 Professional DotNetNuke 4.0

Professional DotNetNuke ASP.NET Portals

## ASP.NET 视频教程系列汇总:

天轰穿ASP.NET2.0 视频教程(全 106 讲, 共七部分)

VS2005 环境下开发ASP.NET 2.0 Web应用程序视屏教程(swf)

[ASP.NET视频]Data 数据访问与操作

[ASP.NET视频]Masterpages 母版页

[ASP.NET视频]Caching 缓存机制

[ASP.NET视频]Contact页

[ASP.NET视频]ASP.NET详细功能介绍

[ASP.NET视频]Localization 本地化

[ASP.NET视频]Membership and Roles management

[ASP.NET视频]Profiles and Themes

[ASP.NET视频]Tips and Tricks

[ASP.NET视频]Web Parts和Personalization详解

## Ajax, ASP.NET Ajax 电子资料汇总:

Ajax基础教程

Ajax宝典

Wrox Beginning Ajax

Ajax in Practice

Ajax模式最佳实践教程

Wrox Professional Rich Internet Applications AJAX and Beyond

O'Reilly Ajax on Java

Practical JavaScript DOM Scripting and Ajax Projects

Creating Web Pages with Asynchronous Javascript and XML

O'Reilly Securing Ajax Applications

Advanced Ajax Architecture and Best Practices

Beginning ASP.NET 2.0 AJAX

Introducing Microsoft ASP.NET AJAX  
Wrox Professional ASP.NET 2.0 AJAX  
ASP.NET AJAX Programmer's Reference





## **Java 语言及其相关开发技术电子资料汇总**

[Java 编程初步 傻瓜书](#)

[数据结与算法 Java语言版](#)

[JSF JavaServer Faces in Action Manning](#)

[Wrox Professional Java JDK 6 Edition](#)

[Java 2 宝典](#)

[侯捷java编程思想 PDF中文版](#)

[Learning Java \(第三版\)](#)

[Beginning Java Programming for Dummies 第二版](#)

[Java 2 核心编程](#)

[Java How to Program \(第六版\)](#)

[Java All-In-One案头参考傻瓜书 \(第二版\)](#)

[21 天自学 Java 6 \(2007 年 5 月更新出版\) PDF](#)

[Java咖啡馆](#)

[Thinking in Java\(第四版\)](#)

[深入学习JFC SWING - Java基础类组件集](#)

[J2EE全实例教程](#)

[Java信息系统设计与开发实例\(第二版\)](#)

[Java优化编程](#)

[Java信息系统设计与开发实例\(第二版\)](#)

[Tricks of the Java Programming](#)

[Wrox Professional Java Native Interfaces with SWT JFace](#)

[Java Swing 第二版 PDF文字版 O'Reilly出版](#)

[O'Reilly - Java Database Programming with JDBC](#)

[JDBC与Java数据库程序设计](#)

[Learning JQurey \(2007 年 7 月最新出版\)](#)

[J2EE设计开发编程指南](#)

[Java Web Services简明教程](#)

[O'Reilly Java and XML \(第二版PDF\)](#)

[O'Reilly Java and XML \(第三版PDF\)](#)

[Java技术XML高级编程](#)

[Expert One-on-One J2EE Design and Development](#)

[Expert One-on-One J2EE Development without EJB](#)

[JBoss - A Developer's Notebook](#)

The Java Programming Language (第四版)  
Spring in Action (第二版)  
Professional Java Development with the Spring Framework  
Core Java Server Faces 第二版  
精通Enterprise JavaBeans  
Enterprise JavaBeans EJB 第四版  
J2EE应用与BEA WebLogic Server (第二版PDF)  
O'Reilly Java Web Services  
Ant权威指南  
Ajax和Java框架高级编程  
Java Web Services简明教程  
Cross-Platform Web Services Using C# and Java  
O'Reilly Ajax on Java  
O'Reilly Java and XSLT  
O'Reilly Java and XML Binding  
O'Reilly Java and SOAP  
Design Patterns Java Companion  
J2EE Java黑客大曝光 开发安全的Java应用程序  
J2ME API 速查手册  
精通J2ME无线编程  
J2ME开发大全  
Java网页开发的艺术  
Java编程高手  
Java 5.0 Tiger程序高手秘笈  
Java2 网络协议技术内幕(附源码)  
Using Enterprise JavaBeans 2  
Java技术实用教程  
企业级Java安全性(构建安全的J2EE应用)  
Java语言集成开发环境Eclipse中文教程  
NetBeans IDE 5.5 企业版高级开发教程  
Using Enterprise JavaBeans 2  
Borland JBuilder Developer's Guide  
Eclipse精要与高级开发技术  
Java 6 3D游戏开发

## 视频教程系列:

J2EE开发IDE Eclipse视频教程 全 9CD 完整版

Java视频教程 孙鑫Java无难事 (全 12CD)

J2EE高级开发视频教程第 01 讲

J2EE高级开发视频教程第 02 讲

J2EE高级开发视频教程第 03 讲

J2EE高级开发视频教程第 04 讲

J2EE高级开发视频教程第 05 讲

J2EE高级开发视频教程第 06 讲

J2EE高级开发视频教程第 07 讲

J2EE高级开发视频教程第 08 讲

J2EE高级开发视频教程第 09 讲

J2EE高级开发视频教程第 10 讲

J2EE高级开发视频教程第 11 讲

## **SQL 语言及各种数据库管理系统电子书汇总**

### **SQL 语言，数据库基础电子资料：**

[SQLite权威指南](#)

[SQL 语法大全中文版](#)

[SQL 语言案头完全参考手册](#)

[SQL - A Practical Introduction](#)

[O'Reilly SQL Tuning](#)

[O'Reilly The Art of SQL](#)

[数据库综合资料库](#)

[数据库设计指南](#)

[Wrox Beginning Database Design](#)

[SQL Puzzles and Answers](#)

[SQL Queries for Mere Mortals](#)

[SQL Puzzles and Answers](#)

[Apress出版 The Berkeley DB Book](#)

[数据库系统概论 浙江大学张军教授主讲\(全 32 讲\)](#)

### **MS SQL Server 电子资料：**

[Transact-SQL Cookbook](#)

[SQL Server 2005 宝典](#)

[Microsoft SQL Server 2005 完全参考](#)

[O'Reilly Learning SQL on SQL Server 2005](#)

[Beginning SQL Server 2005 Programming](#)

[Pro SQL Server 2005 High Availability](#)

[Beginning SQL Server 2005 Administration](#)

[SQL Server 2005 Unleashed](#)

[Pro SQL Server 2005](#)

[A Developer's Guide to SQL Server 2005](#)

[Pro T-SQL 2005 Programmer's Guide](#)

[Beginning Transact-SQL with SQL Server 2000 and 2005](#)

[SQL Server 2005 报表服务](#)

[Wrox Professional SQL Server 2005 Programming](#)

[Scaling Out SQL Server 2005 权威指南](#)



[Sql Server 2005 Performance Optimiztion and Tuning Handbood](#)

[Microsoft SQL Server 2005 编程傻瓜书](#)

[Pro SQL Server 2005 Assemblies](#)

[MS SQL Server 2005 Reporting Essentials](#)

[SQL Server 2005 工具箱内幕](#)

[SQL Server 2005 管理员手册](#)

[SQL Server 2005 工具箱内幕](#)

[SQL Server 2005 数据挖掘](#)

[Pro SQL Server 2005 Service Broker](#)

[Pro SQL Server 2005 Replication](#)

[Sql server 2005 的XML最佳实施策略](#)

[Microsoft SQL Server Black Book](#)

[MS SQL Server2000 宝典](#)

[SQL Server 2000 存储过程和XML编程](#)

[SQL Server 2005 高级数据分析视频教程系列](#)

[SQL Server 2005 盛宴系列视频 全 52 讲](#)

### **MySQL 电子资料:**

[SQL for MySQL Developers](#)

[MySQL教程](#)

[MySQL 5 权威指南\(第三版\)](#)

[MySQL培训经典教程](#)

[MySQL Essential Skills](#)

[MySQL Administrators Guide](#)

[MySQL权威指南 中文版+英文版](#)

[MySQL 4.1.0 中文参考手册](#)

[MySQL in a Nutshell](#)

[Export MySQL](#)

[MySQL and PHP from Scratch](#)

### **其他数据库电子资料:**

[Microsoft Access 2007 初学者指南 2007 年 6 月](#)

[Microsoft Access 2007 宝典](#)

[Microsoft Office Access 2007 VBA宝典](#)

[Wrox出版 Expert Access 2007 Programming](#)

[Access 2007 窗体、报表和查询](#)

[Microsoft Access 2007 数据分析](#)

[Oracle Automatic Storage Management](#)

[Pro Oracle Spatial for Oracle Database 11g](#)

[Oracle 9i 数据库管理员指南](#)

[Wrox Professional Oracle 8i Programming](#)

[O'Reilly Oracle Security](#)

[PL/SQL Study Guide](#)

[Sybase实用教程](#)

[PostgreSQL 对象关系数据库开发](#)

[PostgreSQL 必备参考手册](#)

[PostgreSQL 7 数据库开发指南](#)

[PostgreSQL 8 for Windows 2007 年 3 月最新出版](#)

[Crystal Reports 10 完全参考](#)

[Crystal Reports 10 水晶报表 10 傻瓜书](#)

## HTML、CSS、JavaScript 等 Web 开发技术电子资料汇总

### CSS、HTML、xHTML

#### CSS权威指南

The CSS Anthology (第二版) CSS设计大师设计思路与实践

HTML & XHTML 权威指南(英文CHM版+中文PDF版)

Building a Web Site 傻瓜书

HTML 4 傻瓜书 第五版

css禅意花园 (高级CSS开发)

CSS与DHTML精髓

CSS网页设计师

CSS Hacks and Filter

CSS Mastery高级Web开发标准

CSS 设计艺术

CSS代码效果对比学习

CSS Web Design傻瓜书

Pro CSS Techniques

CSS HTML高级设计模式 Pro CSS and HTML Design Patterns

Build your Own WebSite - The Right Way Using HTML and CSS

Mastering Integrated HTML and CSS

CSS Web Development从入门到精通

Web Publishing with HTML and CSS in One Hour a Day

Designing with Web Standards 网站重构 英文+中文版

Beginning HTML with CSS and XHTML

XHTML Moving toward XML

Building Smart Web 2.0 Applications

How to Do Everything With HTML

Spatial Data on the Web:Modeling and Management

AdvancED DOM Scripting Dynamic Web Design Techniques

精通XHTML程序设计高级编程

XHTML实例精解

XHTML技术内幕

完美HTML设计 - 使用CSS不用Table (第二版)

250 HTML and Web Design Secrets

## JavaScript

JavaScript and Ajax for the Web 第六版

Return on Design

Wrox出版 Silverlight 1.0 (彩页染色代码、全面解析)

Silverlight 1.0 Development with JavaScript

Learn Javascript 高清晰PDF珍藏版

JavaScript 宝典

JavaScript宝典(第六版)

JavaScript实例宝典 PDF文字版

Wrox Beginning JavaScript(第三版)

Professional JavaScript for Web Developers

JavaScript for breakfast

JavaScript in 10 Simple Steps or Less

JavaScript开发人员参考

JavaScript参考大全第二版

JavaScript权威指南(第四版)

JavaScript快速查询手册

Practical JavaScript DOM Scripting and Ajax Projects

JavaScript傻瓜书 第四版

高级Javascript 第二版

Build Your Own Database Driven Website

上百个超酷JS广告代码收集汇总 (第一辑)

上百个超酷JS广告代码收集汇总 (第二辑)

上百个超酷JS广告代码收集汇总 (第三辑)



## **XML 及其相关技术电子书及视频汇总**

[Altova MissionKit 2008 for Enterprise Software Architects完美版](#)

[XML宝典](#)

[Wrox Beginning XML 第四版 2007 出版](#)

[Wrox XML高级编程 2007 版](#)

[Wrox Professional XML Database](#)

[Wrox Professional XML 第二版](#)

[无废话XML](#)

[XAML简明教程 CHM+PDF版](#)

[XML傻瓜书 第四版 PDF文字版](#)

[XML宝典\(第二版\) PDG](#)

[XML编程从入门到精通](#)

[O'Reilly Learning XML](#)

[O'Reilly Learning XML \(第二版\)](#)

[XML终极教程 PDF版](#)

[XML轻松学习手册](#)

[XML Pocket Reference \(第二版\)](#)

[XML编程](#)

[XML in a Nutshell \(第二版\)](#)

[XML 21 天自学教程\(第三版\)](#)

[实战XML\(第二版\)](#)

[轻松搞定XML](#)

[XML实用大全](#)

[XML解决方案开发实务](#)

[XSLT Quickly](#)

[XML参考手册\(第 4 版\)](#)

[XML in Theory and Practice](#)

[Real World Xml Web Services](#)

[Practical Transformation with XSLT and XPath](#)

[Navigating XML With XPath 1.0/2.0 Kick Start](#)

[O'Reilly XSLT Cookbook](#)

[XML之使用SOAP开发Web服务](#)

[XML编程 XQuery专家教程](#)

[XPath,XLink,XPointer,and XML学习手册](#)

O'Reilly XSLT

Beginning XSLT 2.0 从入门到精通

XML学习系列之 XSL-FO权威指南

XML,XSLT,Java and JSP

SVG编程指南

XML Security

Beginning RSS and Atom Programming

O'Reilly XQuery 2007 年最新版

孙鑫xml视频教程 全三讲

## **PHP 语言、Apache 相关电子书及视频下载汇总**

[PHP Designer 2008 专业版+特别文件 完美版](#)

[PHP Designer 2007 专业版+特别文件 完美版](#)

[Practical Web 2.0 Applications with PHP\(Apress 2008 最新版\)](#)

[Practical Apache Struts2 Web 2.0 Projects](#)

[O'Reilly Learning PHP & MySQL 第二版](#)

[PHP 5 傻瓜书](#)

[Wrox Beginning PHP 5](#)

[PHP 5 Advanced](#)

[PHP 5 与MySQL编程初学者指南](#)

[PHP 5 和MySQL 5 从入门到精通 PDF文字版](#)

[PHP 4.1 从入门到精通](#)

[PHP技术内幕](#)

[PHP最新参考手册](#)

[PHP程序设计](#)

[PHP经典 100 例](#)

[Object Oriented PHP Concepts Techniques and Code](#)

[The PHP Anthology 第二版](#)

[PHP API使用完全指南](#)

[Wiley出版 Making Use of PHP](#)

[PHP实例教程](#)

[PHP in Action](#)

[PHP+MySQL网络开发技术](#)

[Dreamweaver CS3 with CSS, Ajax, and PHP](#)

[AJAX and PHP Building Responsive Web Applications](#)

[Beginning Ajax with PHP](#)

[PHP Programming with PEAR](#)

[PHP MySQL and Apache自学教程](#)

[PHP Apache和MySQL网页开发初步](#)

[24 小时学会使用PHP MySQL Apache](#)

[PHP MySQL 网络应用程序开发核心](#)

[Professional LAMP - Linux,Apache,MySQL and PHP 5 Web Development](#)

[Setting Up LAMP - Getting Linux Apache MySQL and PHP Working Together](#)

[Beginning PHP,Apache,MySQL Web Developmnet](#)

PHP Data Objects for MySQL

MySQL and PHP from Scratch

Extending and Embedding PHP

Wiley出版 Secure PHP Development

O'Reilly Building Tag Clouds in Perl and PHP

Wrox Professional Apache Tomcat 5

PHP MySQL编程初学者指南

Beginning PHP and Oracle (PDF文字版)

Advanced PHP for Web Professionals

Apache Server 2.0 实用指南

Apache管理员手册

Apache使用指南与实现原理

PHP专业项目实例开发 中文PDF版

PHP高级开发技术与实例 中文PDF影版

PHP 5 for Flash

O'Reilly - Tomcat权威指南

Wrox Professional Apache Tomcat 6

Pro Jakarta Tomcat 5

Foundations of PEAR - Rapid PHP Development

黑客基地PHP开发中级提高班 PHP视频教程 全 11 讲完整版

一周学会PHP编程 台湾中原大学PHP教程 第一讲

一周学会PHP编程 台湾中原大学PHP教程 第二讲

一周学会PHP编程 台湾中原大学PHP教程 第三讲

一周学会PHP编程 台湾中原大学PHP教程 第四讲

一周学会PHP编程 台湾中原大学PHP教程 第五讲



## 上百个 **Linux**、**BSD**、**Unix** 学习电子书+视频下载汇总

### Linux 学习必备系列之 Linux 各发行版本资料汇总

[Linux宝典 2007 版](#)

[Linux宝典 2005 版](#)

[Beginning Ubuntu Linux](#)

[Ubuntu Linux宝典](#)

[Ubuntu部落](#)

[Ubuntu中文Wiki离线PDF版](#)

[Ubuntu Unleashed](#)

[Ubuntu Linux for Non-Geeks 第一版](#)

[Ubuntu Linux for Non Geeks 第二版](#)

[Moving to Ubuntu Linux](#)

[O'Reilly Ubuntu Hacks](#)

[Debian GNU Linux安装与基本配置](#)

[Debian GNU/Linux宝典](#)

[O'Reilly Learning Debian GNU/Linux](#)

[Redhat Linux 学习指南 第二版](#)

[Learning Red Hat Linux 第三版](#)

[Red Hat Linux 9 魔鬼式培训教程](#)

[Redhat Linux 9 从入门到精通](#)

[Fedora 6 and Red Hat Enterprise Linux宝典](#)

[Red Hat Linux网络管理工具](#)

[Red Hat Fedora Linux宝典](#)

[Red Hat Linux Fedora24 小时自学教程](#)

[Red Hat Linux Fedora for Dummies](#)

[Red Hat Fedora Linux 2 案头完全参考傻瓜书](#)

[红帽企业版Linux国际化支持语配置指南](#)

[红帽企业版 5.0 快速布置指南](#)

[红帽企业Linux安装指南](#)

[RedHat Linux AS 4 平台下Oracle 9](#)

[Red Hat Enterprise Linux 4 傻瓜书](#)

[Fedora 7 Unleashed](#)

[Redhat Fedora core 6 unleashed](#)

[Fedora Core 5 初学者指南](#)

[Redflag HA Cluster 4.1 完全参考](#)

[Redflag Linux Server 4.0 用户手册](#)

[Redflag Data Center 5.0 系统管理](#)

[Redflag Linux Desktop 5 用户手册](#)

[Freebsd简明教程](#)

[FreeBSD 6 Unleased](#)

[FreeBSD使用大全\(第二版\)](#)

[FreeBSD完全手册\(第三版\)](#)

[FreeBSD Handbook PDF 中文版+英文版](#)

[BSD FreeBSD Architecture Handbook](#)

[Absolute BSD - The Ultimate Guide to FreeBSD](#)

[Designing BSD Rootkits - An Introduction to Kernel Hacking](#)

[FreeBSD 6.0 架设管理与应用](#)

[Beginning SUSE Linux 第二版](#)

[SuSe Linux](#)

[SuSe Linux初学者从入门到精通](#)

[SuSe Linux 10 宝典](#)

[SuSe Linux 10 完全参考](#)

[SuSe Linux 10 傻瓜书](#)

[Suse Linux 10 新手指南](#)

[SUSE Linux 企业服务器权威指南](#)

[Suse linux 9.3 用户手册](#)

[Suse linux 9.3 管理员手册](#)

[Knoppix Hacks](#)

## **Linux 学习必备系列之 基础应用资料汇总**

[Linux新手管理指南](#)

[Linux是如何工作的](#)

[Linux简明教程 第四版](#)

[Linux傻瓜书 第六版](#)

[Linux All-In-One Desk Reference for Dummies 2006 版](#)

[Linux All-In-One Desk Reference for Dummies 2005 版](#)

[Learning the Vi Editor \(第六版\)](#)

[GNU Emacs 参考手册](#)

[Linux Cookbook](#)  
[Linux for Non-Geeks](#)  
[Linux文件查找命令find、xargs详述](#)  
[Linux Desktop Hacks](#)  
[LINUX 24 学时教程](#)  
[Linux案头参考\(第二版\)](#)  
[O'Reilly Linux简明教程 第五版](#)  
[Linux系统管理白皮书](#)  
[Linux系统一本通](#)  
[Linux实用培训学习教程](#)  
[Linux命令参考大全](#)  
[送给初学Linux的穷人Linux系统指令大全](#)  
[Linux命令完全参考](#)  
[Linux命令字典](#)  
[Linux Complete Command Reference](#)  
[Linux故障排除宝典](#)  
[Linux桌面系统提速法宝](#)  
[从Windows转向Linux基础教程](#)  
[让Linux像Windows一样方便](#)  
[Learning the UNIX Operating System 第四版](#)  
[UNIX和Linux权威教程 \(第三版\)](#)  
[Unix完全参考\(第二版\)](#)  
[Unix傻瓜书](#)  
[Unix简明教程 第四版](#)  
[Unix教程网络篇](#)  
[UNIX for OpenVMS Users 第三版](#)  
[Linux网络管理员手册](#)

## **Linux 学习必备系列之 高级应用资料汇总**

[A Beginner's Guide to LVM](#)  
[Novell出版 Linux防火墙 第三版 chm格式](#)  
[No Starch出版 Linux Firewalls](#)  
[2 小时玩转iptables企业版](#)  
[Securing Optimizing Linux The Hacks Solution](#)  
[SUSE Linux 企业服务器权威指南](#)

[Red Hat Linux网络管理工具](#)

[Linux进程管理教程](#)

[Linux下安装Oracle完全参考](#)

[RedHat Linux AS 4 平台下Oracle 9](#)

[Redflag HA Cluster 4.1 完全参考](#)

[Hack Proofing Linux](#)

[Linux Server Hacks](#)

[Understanding Linux Network Internals](#)

[Linux Power Tools](#)

[Linux On The Mainframe](#)

[Linux Network Servers](#)

[O'Reilly Building Embedded Linux Systems](#)

[Advanced Linux Networking](#)

[Linux Security Cookbook](#)

[保护Linux系统 - Linux安全生存指南](#)

[SELinux NSAs - Open Source Security Enhanced Linux](#)

[Hardening Linux](#)

[Building Secure Servers With Linux](#)

[详细剖析Linux和Unix两系统病毒威胁](#)

[Linux黑客大曝光 - Linux安全机密与解决方案](#)

[使用Ipfilter建立FreeBSD加固防火墙](#)

[Mastering FreeBSD and OpenBSD Security](#)

[User Mode Linux](#)

[Linux Appliance Design](#)

[Linux Device Drivers](#)

[Multitool Linux Practical Uses for Open Source Software](#)

[高性能Linux集群](#)

[Linux网络构架设计与实现](#)

[Building Embedded Linux Systems](#)

[Unix Linux管理自动化](#)

[Linux Power Tools](#)

[Building Applications with the Linux Standard Base](#)

[Hacking Linux Exposed](#)

[Building Secure Servers with Linux](#)

[SSH - Unix Secure Shel tool](#)



[UNIX 系统安全工具](#)

[Unix for Oracle DBAs Pocket Reference](#)

[Unix 网络安全实用教程](#)

[Unix备份与恢复全攻略](#)

[UNIX Power Tools \(第三版\)](#)

[Practical Unix and Internet Security \(第三版\)](#)

**Linux 学习必备系列之 Linux 环境编程资料汇总**

[Understanding The Linux Kernel 第一版](#)

[Understanding The Linux Kernel 第二版](#)

[Understanding The Linux Kernel 第三版](#)

[Linux内核精要](#)

[Understanding the Linux Kernel - 理解Linux内核](#)

[Linux内核源代码情景分析 中文版 \(上下册\)](#)

[O'Reilly Bash Cookbook](#)

[101 个超酷Shell脚本](#)

[Bash快速参考](#)

[Bash Beginners Guide](#)

[Perl入门及高级编程](#)

[Perl语言编程](#)

[Perl指南](#)

[Perl编程思想](#)

[O'Reilly 精通Perl编程](#)

[O'Reilly Perl and XML](#)

[Perl 5 21 天自学教程](#)

[轻松学习Linux编程](#)

[Linux应用开发基础](#)

[Linux 网络编程](#)

[Python简明教程](#)

[Unix编程艺术 The Art of Unix Programming](#)

[Linux编程白皮书](#)

[A Practical Guide to Linux Commands Editors and Shell Programming](#)

[Linux案头参考\(第二版\)](#)

[Linux与Unix Shell编程指南](#)

[Linux Shell Scripting with Bash](#)

Unix Shell Programming(第三版)  
Linux Debugging And Performance Tuning  
Linux系统分析与高级编程技术  
十分钟Unix自学教程 第二版  
Korn Shell:Unix and Linux Programming Manual  
Unix shell范例教程 (第四版)  
Unix环境高级编程  
Unix环境高级编程 第二版  
Unix Systems Programming  
Linux编程从入门到精通  
Linux实例编程  
Linux环境编程 GCC完全参考  
Linux应用程序开发指南 使用Gtk+ Gnome库  
Linux C高级程序员指南  
Sams Mono Kick Start - Linux环境的.NET编程

### **Linux 学习必备系列之 Linux 视频教程系列**

Redhat认证 RHCE视频教程 全七部分 高清AVI格式  
楚广明 24 小时学通Linux 第一讲 基础安装与GNU历史  
楚广明 24 小时学通Linux 第二讲 基本概念与命令  
楚广明 24 小时学通Linux 第三讲 系统设置  
楚广明 24 小时学通Linux 第四讲  
楚广明 24 小时学通Linux 第五讲 DNS服务器  
楚广明 24 小时学通Linux 第六讲  
楚广明 24 小时学通Linux 第七讲 Apache与Tomcat整合实验  
SUSE Linux Enterprise 10 精妙解决方案视频教程  
FreeBSD 6.2 服务器架设视频教程 (全五部分)  
高效架设Redhat Linux服务器全过程视频教程  
楚广明主讲 FreeBSD视频教程 swf版  
红旗Linux安装全过程视频教程  
debian etch安装全程攻略视频  
Arch Linux安装全程攻略视频  
Linux基础教程视频(全四讲)  
Linux从入门到精通视频教程(swf版) 基础安装配置  
Linux从入门到精通视频教程(swf版) 服务器配置

[Linux从入门到精通视频教程\(swf版\) 安全配置](#)  
[Linux下访问NTFS分区配置全过程视频教程](#)  
[黑客基地Linux视频教程系列之linux简介](#)  
[黑客基地Linux视频教程系列之文件系统](#)  
[黑客基地Linux视频教程系列之在字符界面下安装](#)  
[黑客基地Linux视频教程系列之图形界面安装linux并配置](#)  
[黑客基地Linux视频教程系列之体验linux单操作系统的安装](#)  
[黑客基地Linux视频教程系列之linux与windows双系统安装](#)  
[黑客基地Linux视频教程系列之GRUB的配置方法](#)  
[黑客基地Linux视频教程系列之GRUB相关问题解决方法](#)  
[黑客基地Linux视频教程系列之桌面简介与网络配置](#)  
[黑客基地Linux视频教程系列之VI编辑器使用](#)  
[黑客基地Linux视频教程系列之linux中的用户管理](#)  
[黑客基地Linux视频教程系列之文件系统常用命令](#)  
[黑客基地Linux视频教程系列之网络基础配置](#)  
[黑客基地Linux视频教程系列之软件包安装使用](#)  
[黑客基地Linux视频教程系列之目录文件的操作命令](#)  
[黑客基地Linux视频教程系列之linux下的日志文件](#)  
[黑客基地Linux视频教程系列之linux中的进程管理](#)  
[黑客基地Linux视频教程系列之linux下压缩文件使用教程](#)  
[黑客基地Linux视频教程系列之shell编程简介](#)  
[黑客基地Linux视频教程系列之DHCP服务器](#)  
[黑客基地Linux视频教程系列之samba服务器\(二讲\)](#)  
[黑客基地Linux视频教程系列之DNS服务器设置详解\(三讲\)](#)  
[黑客基地Linux视频教程系列之linux下架设apache服务器\(四讲\)](#)  
[黑客基地Linux视频教程系列之linux下简单的网络安全](#)  
[黑客基地Linux视频教程系列之iptables防火墙简单使用](#)  
[黑客基地Linux视频教程系列之linux内核与总结](#)  
[边学边用linux视频教程 第 01 讲\(全 24 讲\)](#)  
[边学边用linux视频教程 第 02 讲\(全 24 讲\)](#)  
[边学边用linux视频教程 第 03 讲\(全 24 讲\)](#)  
[边学边用linux视频教程 第 04 讲\(全 24 讲\)](#)  
[边学边用linux视频教程 第 05 讲\(全 24 讲\)](#)  
[边学边用linux视频教程 第 06 讲\(全 24 讲\)](#)  
[边学边用linux视频教程 第 07 讲\(全 24 讲\)](#)

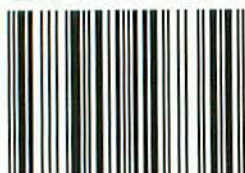
边学边用linux视频教程 第 08 讲(全 24 讲)  
边学边用linux视频教程 第 09 讲(全 24 讲)  
边学边用linux视频教程 第 10 讲(全 24 讲)  
边学边用linux视频教程 第 11 讲(全 24 讲)  
边学边用linux视频教程 第 12 讲(全 24 讲)  
边学边用linux视频教程 第 13 讲(全 24 讲)  
边学边用linux视频教程 第 14 讲(全 24 讲)  
边学边用linux视频教程 第 15 讲(全 24 讲)  
边学边用linux视频教程 第 16 讲(全 24 讲)  
边学边用linux视频教程 第 17 讲(全 24 讲)  
边学边用linux视频教程 第 18 讲(全 24 讲)  
边学边用linux视频教程 第 19 讲(全 24 讲)  
边学边用linux视频教程 第 20 讲(全 24 讲)  
边学边用linux视频教程 第 21 讲(全 24 讲)  
边学边用linux视频教程 第 22 讲(全 24 讲)  
边学边用linux视频教程 第 23 讲(全 24 讲)  
边学边用linux视频教程 第 24 讲(全 24 讲)



# 全国计算机技术与软件专业技术资格（水平）考试辅导用书

根据人事部、信息产业部文件，计算机技术与软件专业技术资格（水平）考试纳入全国专业技术人员职业资格证书制度的统一规划。通过考试获得证书的人员，表明其已具备从事相应专业岗位工作的水平和能力，用人单位可根据工作需要从获得证书的人员中择优聘任相应专业技术职务（技术员、助理工程师、工程师、高级工程师）。计算机技术与软件专业实施全国统一考试后，不再进行相应专业技术职务任职资格的评审工作。

ISBN 7-302-10735-1



9 787302 107354 >

定价：46.00元

[www.TopSage.com](http://www.TopSage.com)